

UNIVERSITY OF STELLENBOSCH
DEPARTMENT OF INDUSTRIAL ENGINEERING

AN INTEGRATED CONTROL SYSTEM FOR THE SUBSETS OF A MOBILE ROBOT

THESIS PRESENTED BY *Ms LM GERTENBACH* IN PARTIAL
FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE
OF MASTER OF ENGINEERING AT THE UNIVERSITY OF
STELLENBOSCH.

Study Leader: Mr CJ Fourie

Department of Industrial Engineering

December 2001

DECLARATION

I, the undersigned hereby declare that the work contained in this thesis is my own original work and has not previously in its entirety or in part been submitted at any university for a degree.

Ms LM Gertenbach

SYNOPSIS

This project, titled *An Integrated Control System for the Subsets of a Mobile Robot*, concerns the integration of subsets for a mobile robotic system developed in the past. It was found that these subsets would not integrate with ease and that integration was not taken into account when the subsets were developed. The complete problem statement is given in chapter one.

Before trying to solve the problem the author did a literature review to equip her with the necessary knowledge and skills needed to solve the problem. The literature reviewed included topics like Computer Integrated Manufacturing (CIM) and Mobile Robotic. A thorough study of the developed subsets was also completed.

The proposed control system is software-based and developed in three programming languages:

- TURBO PASCAL,
- BORLAND DELPHI, and
- MATLAB.

The DELPHI and MATLAB programs run on the base station. This computer communicates with the MOBILE ROBOT used in this project (MOBROB) using RF communication hardware and software. This software is part of the DELPHI programs. The DELPHI program is the starting point. This program then calls the MATLAB programs or sends data to the robot. It is important to set the hardware as prompted.

The process basically consists of creating path files on the base station. These files are then sent to the robot. The robot then drives to a destination, updates its position and sends the new position back to the base station.

The onboard programs used by the robot were developed in TURBO PASCAL. These programs are responsible for the control of the robot's hardware. The control system is discussed in detail in chapters five, six and seven.

The project was completed successfully and all the objectives set at the beginning of the project were met. These objectives are given in chapter one.

This document consists of four sections. The first section is the introduction, the second section the literature review, the third section discusses the control system and the fourth section discusses the results and contains conclusions and recommendations.

APPENDIXES A to F gives mathematical calculations, user guides and computer programming codes.

OORSIG

Die projek, getiteld *'n Geïntegreerde beheerstelsel vir die subdele van 'n Mobiele Robot*, handel oor die integrasie van voorafontwikkelde dele vir 'n mobiele robotiese stelsel. Daar is gevind dat die dele nie maklik is om te integreer nie en dat integrasie glad nie in ag geneem is toe die dele ontwikkel is nie. Die volledige probleemstelling word in hoofstuk een gegee.

Voordat die probleem aangepak is, is 'n literatuurstudie voltooi om die skrywer toe te rus met die kennis wat nodig is vir die oplos van die probleem. Die literatuurstudie het gehandel oor Rekenaar-geïntegreerde vervaardiging (CIM), mobiele robotte en die verskillende voorafontwikkelde dele van die stelsel. Die literatuurstudie vorm hoofstukke twee, drie en vier van hierdie dokument.

Die voorgestelde beheerstelsel is sagteware gebaseer en is in drie rekenaar programmerings-tale ontwikkel:

- TURBO PASCAL,
- BORLAND DELPHI en
- MATLAB.

Die DELPHI and MATLAB programme loop op die beheerstasie rekenaar. Die rekenaar kommunikeer met die robot deur middel van RF kommunikasie hardeware en sagteware. Hierdie sagteware vorm deel van die DELPHI sagteware. Daar word begin by die Delphi program. Die program roep dan die MATLAB programme op of stuur data na die robot toe. Dit is belangrik om die apparatuur op te stel soos in die dokument aangeteken is, voor die programme geloop word.

Die proses bestaan basies uit die skep van pad-lêers op die beheerstasie. Hierdie lêers word dan na die robot gestuur, wat van sy huidige posisie na 'n bestemming toe beweeg. Wanneer die taak voltooi is, word die nuwe posisie van die robot teruggestuur na die beheerstasie.

Op die robot self, loop die PASCAL program, wat verantwoordelik is vir die beheer van die robot hardeware. Die beheerstelsel word deeglik bespreek in hoofstukke vyf, ses en sewe.

Die projek is suksesvol afgehandel en daar word aan al die doelstellings wat aan die begin van die projek daar gestel is, voldoen. Hierdie doelstellings word in hoofstuk een gegee.

Die verslag bestaan uit vier dele. Die eerste deel is 'n inleiding, die tweede deel 'n literatuurstudie, die derde deel bespreek die beheerstelsel en die vierde deel is 'n samevatting waar die resultate bespreek word en gevolgtrekkings en aanbevelings gemaak word.

Daarna volg 'n aantal bylae met wiskundige afleidings, gebruikersgidse en rekenaar program kodes.

ACKNOWLEDGEMENTS

The author wants to thank the following people for their huge contribution to this thesis.

- ☞ Mr Fourie for his guidance and support during the project.
- ☞ G-J van Rooyen for his assistance with MATLAB.
- ☞ Wouter Brand for his patience and support.
- ☞ Ms Oosthuizen for reviewing the grammar of this document.

TABLE OF CONTENTS

Declaration.....	ii
Synopsis.....	iii
Oorsig.....	v
Acknowledgements.....	vii
Table of Contents.....	viii
List of Figures.....	xv
List of Tables	xviii
Glossary	xix
SECTION ONE: Introduction.....	1
1 Introduction.....	2
1.1 PROBLEM STATEMENT	3
1.2 THESIS OBJECTIVES.....	4
1.3 LIMITATIONS	5
1.4 STRUCTURE OF THE THESIS.....	5
SECTION TWO: Literature Review.....	7
2 Computer Integrated Manufacturing (CIM).....	8
2.1 INTRODUCTION.....	8
2.2 SKILLS REQUIRED FOR THE SUCCESSFUL IMPLEMENTATION OF A CIM SYSTEM	10
2.3 COMPLEXITY OF SYSTEMS	11
2.4 CIM WHEEL.....	13
2.5 BENEFITS OF CIM	13
2.6 TRENDS CONCERNING CIM	15

2.7	SUMMARY	16
3	Literature Review on Mobile Robots	17
3.1	MOBILE ROBOT DESIGN	17
3.1.1	Steering Arrangements	17
3.1.2	Shape and Size.....	19
3.1.3	Power Source	20
3.1.4	Driving Source	20
3.1.5	Control Computer.....	20
3.1.6	Software-Hardware Interface.....	21
3.2	MOBILE ROBOT POSITIONING	22
3.2.1	Odometry.....	22
3.2.2	Inertial Navigation.....	24
3.2.3	Magnetic Compasses	24
3.2.4	Active Beacons	25
3.2.5	Global Positioning System	26
3.2.6	Landmark Navigation.....	26
3.2.7	Map-Based Positioning.....	28
3.3	MAPPING AND PATH-PLANNING.....	28
3.3.1	Operating with full a priori information	29
3.3.2	Operating with no a priori knowledge	29
3.3.3	Mapping	29
3.3.4	Path-planning Algorithms	32
3.4	COMMUNICATION WITH MOBILE ROBOTS.....	35
3.5	DOCKING TECHNIQUES FOR MOBILE ROBOTS	35
3.5.1	Infrared Triangulation	36
3.5.2	Wire-Guided Docking.....	36
3.5.3	Magnetic Sensors	37
3.5.4	Infrared Active Beacon.....	37
3.5.5	Magnetic Reed Switches.....	37
3.6	FUZZY LOGIC CONTROL	38
3.6.1	Basic Concepts of Fuzzy Sets and Fuzzy Logic	38
3.6.2	The Basic Architecture of Fuzzy Logic Controllers.....	40
3.6.3	Four Important Issues	42
4	The MOBROB Project.....	43
4.1	FUZZY LOGIC CONTROLLER FOR AN AUTONOMOUS GUIDED VEHICLE (DEIST, 1993)	43
4.2	THE DESIGN AND CONSTRUCTION OF A MOBILE ROBOT FOR MATERIAL HANDLING (NEL, 1997)	45

4.2.1	<i>MOBROB Hardware</i>	46
4.2.2	<i>MOBROB Software</i>	46
4.3	THE MAPPING AND PATH-PLANNING OF AN INDUSTRIAL MOBILE ROBOT (VAN ROOYEN, 1997) 46	
4.3.1	<i>Optical Robotic Potential Field Mapping</i>	46
4.3.2	<i>Presentation of the Factory</i>	47
4.3.3	<i>Field associated with charges</i>	48
4.4	A STUDY OF THE DIFFERENT NAVIGATIONAL TECHNIQUES FOR A MOBILE ROBOT (MENTZ, 1997)	49
4.4.1	<i>Detailed Description of the navigational method developed by Mentz</i>	49
4.4.2	<i>The Direction the Robot is Facing</i>	53
4.4.3	<i>Parts of the System</i>	55
4.5	COMMUNICATION WITH A MOBILE ROBOT USING RADIO FREQUENCIES (STEENKAMP, 1998) 57	
4.5.1	<i>Communication Hardware</i>	57
4.5.2	<i>Communication Software</i>	60
5	Literature Conclusions	62
5.1	GENERAL CONCLUSIONS	62
5.2	COMPUTER INTEGRATED MANUFACTURING	62
5.3	MOBILE ROBOTICS	62
5.4	THE AS-IS MOBROB PROJECT	62
5.5	GENERAL REMARKS	63
SECTION THREE: The Integrated Control System and its Components		64
6	The Integrated Control System	65
6.1	GENERAL	65
6.2	OVERVIEW OF THE DELPHI-BASED PROGRAMS	68
6.3	OVERVIEW OF THE MATLAB-BASED PROGRAMS	70
6.3.1	<i>Overview of the Quick Path-planning Program</i>	70
6.3.2	<i>Overview of the Path-planning Program</i>	70
6.4	OVERVIEW OF THE PASCAL-BASED PROGRAMS	73
6.5	SUMMARY	75
7	MOBROB Hardware	76
7.1	MOBROB – THE ROBOT	76

7.2	MOBROB – THE RF UNIT	77
7.3	MOBROB – THE BASE STATION	78
8	MOBROB Software	79
8.1	GENERAL	79
8.1.1	<i>BORLAND DELPHI's Abilities</i>	79
8.1.2	<i>MATLAB's Abilities</i>	79
8.1.3	<i>TURBO PASCAL's Abilities</i>	80
8.2	DELPHI-BASED PROGRAMS	80
8.3	MATLAB-BASED PROGRAMS	81
8.4	PASCAL-BASED PROGRAMS	85
8.5	DIFFERENT FILE FORMATS	86
9	Control System Conclusions	91
9.1	GENERAL	91
9.2	THE INTEGRATED CONTROL SYSTEM	91
9.3	THE MOBROB HARDWARE	91
9.4	THE MOBROB SOFTWARE	92
	SECTION FOUR: Validation	93
10	Discussion of Results	94
10.1	STATION CONTROL	94
10.2	CREATING A NEW MAP	94
10.3	EDITING AN EXISTING MAP	97
10.4	PATH-PLANNING	99
10.5	QUICK PATH-PLANNING	100
10.6	SEND A PATH TO MOBROB	101
10.7	SEND A QUICK-PLANNED PATH TO MOBROB	101
10.8	MOBROB DRIVES TO DESTINATION	102
10.9	EVALUATION OF RESULTS	104
10.9.1	<i>Advantages of the System</i>	104
10.9.2	<i>Disadvantages of the System</i>	104

11	Final Conclusions	105
11.1	GENERAL	105
11.2	COMPUTER INTEGRATED MANUFACTURING	105
11.3	THE EXISTING MOBROB PROJECT.....	105
11.4	THE INTEGRATED CONTROL SYSTEM.....	106
11.5	FINAL REMARKS.....	106
12	Recommendations	107
12.1	GENERAL	107
12.2	CONCURRENT INTEGRATION	107
12.3	MOBROB – THE ROBOT	107
13	Summary.....	108
13.1	LITERATURE REVIEW	108
13.2	THE INTEGRATED CONTROL SYSTEM.....	109
13.3	THE FUTURE OF MOBROB.....	110
14	References.....	111
15	Bibliography	114
APPENDIX A: MOBROB Subset Synopsis by the different authors		I
COMMUNICATION WITH A MOBILE ROBOT USING RADIO FREQUENCIES (STEENKAMP, 1998)		I
THE MAPPING AND PATH-PLANNING OF AN INDUSTRIAL MOBILE ROBOT (VAN ROOYEN, 1997)...		I
A STUDY OF THE DIFFERENT NAVIGATIONAL TECHNIQUES FOR A MOBILE ROBOT (MENTZ, 1997)		II
THE DESIGN AND CONSTRUCTION OF A MOBILE ROBOT FOR MATERIAL HANDLING (NEL, 1997)		III
APPENDIX B: Mathematical Calculations		V
MATHEMATICAL CALCULATIONS FOR THE GLOBAL NAVIGATIONAL SYSTEM.....		V
APPENDIX C: MOBROB User Guide		X
REQUIRED FILES		X
15.1.1	DELPHI Files	X

15.1.2	<i>MATLAB Files</i>	X
15.1.3	<i>PASCAL Files</i>	XI
DIRECTORY STRUCTURE		XI
15.1.4	<i>Base Station</i>	XII
15.1.5	<i>Onboard Computer</i>	XII
MOBROB WIRING.....		XIII
WALKTHROUGH		XIV
APPENDIX D: MOBROB Pascal Programs		XVI
INTCTRL4.PAS.....		XVI
UNIT MAP		XXXVII
UNIT NAVDRIVE		LIII
UNIT RFU309		LVI
APPENDIX E: MOBROB DELPHI Programs		LXVII
FILESEND.PAS		LXVII
ABOUT.PAS		LXXXIII
REMINDERRECEIVE.PAS.....		LXXXV
REMINDERSEND.PAS		LXXXVI
APPENDIX F: MOBROB MATLAB Programs		LXXXVIII
MAIN.M		LXXXVIII
MAINQP.M		LXXXVIII
CREAM.M.....		LXXXIX
CREATMAP.M.....		XC
EDITMAP.M.....		XCII
INISLIDE.M.....		XCIV
LOADMAP.M		XCIV
LOADMAPQP.M		XCv
LOADPOS.M.....		XCvI
NEWMAP.M		XCvI
NEWPOS.M		XCvII

POSIB.M.....XCIX

POTMAP.M C

POTMAP2.M CI

STARGET.M CII

STARGETQP.M CII

WALK.M CIII

WALKQP.M CXII

LIST OF FIGURES

FIGURE 1.1: THESIS PROBLEM PLACED IN PERSPECTIVE (WITH THE MOBROB PROJECT)	4
FIGURE 1.2: DIFFERENT SECTIONS OF THIS DOCUMENT	5
FIGURE 2.1: A CIM SYSTEM CONCEPT (RÁNKY (1986))	10
FIGURE 2.2: THE VARIETY OF A SYSTEM (WALDNER (1990))	12
FIGURE 2.3: THE VARIETY OF A COMBINATION OF SYSTEMS (WALDNER (1990)).....	12
FIGURE 2.4: THE SME CIM WHEEL © 1985, SOCIETY OF MANUFACTURING ENGINEERS, DEARBORN, MI 48121	15
FIGURE 3.1: DIFFERENTIAL STEERING ARRANGEMENT	18
FIGURE 3.2: CASTORS CAUSING WHEEL SLIPPAGE	18
FIGURE 3.3: CAR AND TRICYCLE TYPE DRIVES	19
FIGURE 3.4: SYNCHRO DRIVE.....	19
FIGURE 3.5: GREEDY MAPPING (KOENIG ET AL, 2001)	31
FIGURE 3.6: EXPANSION OF AN OBSTACLE AND SHRINKAGE OF THE ROBOT.....	33
FIGURE 3.7: DISCRETE GROWING OF AN OBSTACLE	33
FIGURE 3.8: DISTANCE MEMBERSHIP FUNCTIONS.....	39
FIGURE 3.9: THE BASIC ARCHITECTURE OF A FUZZY LOGIC CONTROLLER.....	40
FIGURE 4.1: PICTURE OF MOBROB.....	43
FIGURE 4.2: END VIEW OF A MODEL AGV (NEL, 1997).....	44
FIGURE 4.3: A 3D PLOT OF ELECTRIC FIELD LINES	49
FIGURE 4.4: BASIC CONFIGURATION OF THE SYSTEM	51
FIGURE 4.5: PARAMETERS USED IN CALCULATIONS.....	52

FIGURE 4.6: SCENARIO 0	53
FIGURE 4.7: SUB-SCENARIO 0.1 FOR DIRECTION OF FACING	54
FIGURE 4.8: SUB-SCENARIO 0.2 OR DIRECTION OF FACING	54
FIGURE 4.9: THE TRANSMITTING PART OF THE SYSTEM (MENTZ, 1997)	55
FIGURE 4.10: THE RECEIVING PART OF THE SYSTEM (MENTZ, 1997)	56
FIGURE 4.11: TYPICAL MAXIM232 CIRCUIT DIAGRAM.....	59
FIGURE 4.12: 5V REGULATOR.....	60
FIGURE 6.1: FLOW DIAGRAM OF CONTROL SYSTEM	67
FIGURE 6.2: FLOW DIAGRAM OF DELPHI-BASED BASE STATION CONTROL PROGRAM.....	69
FIGURE 6.3: FLOW DIAGRAM FOR THE MATLAB-BASED QUICK PATH-PLANNING PROGRAM.....	71
FIGURE 6.4: FLOW DIAGRAM FOR MATLAB-BASED PATH-PLANNING PROGRAM.....	72
FIGURE 6.5: FLOW DIAGRAM OF PASCAL-BASED PROGRAM.....	74
FIGURE 7.1: BASE STATION RF COMMUNICATION UNIT CIRCUIT DIAGRAM	77
FIGURE 7.2: ONBOARD RF COMMUNICATION UNIT CIRCUIT DIAGRAM.....	78
FIGURE 8.1: USER INTERFACE CREATED BY THE MAIN PROCEDURE	83
FIGURE 8.2: THE USER DEFINES THE ROOM DIMENSIONS	83
FIGURE 8.3: SELECT A MATLAB MAP FILE TO LOAD	84
FIGURE 8.4: THE USER ENTERS A FILENAME TO SAVE THE CREATED PATH.....	84
FIGURE 8.5: DIFFERENT DIRECTION SYSTEMS USED BY MATLAB PATH-PLANNING AND PASCAL CONTROL	87
FIGURE 8.6: *.MAP FILE USED BY THE PASCAL-BASED CONTROL PROGRAM	88
FIGURE 8.7: *.PTH FILE USED BY THE PASCAL-BASED CONTROL PROGRAM	89
FIGURE 8.8: MATLAB MAP FILE.....	89
FIGURE 8.9: MATLAB POTENTIAL FIELD FILE	90

FIGURE 8.10: CURRENT.POS FILE USED BY BOTH THE CONTROL PROGRAM ON MOBROB AND THE MATLAB PATH-PLANNING PROGRAM.....	90
FIGURE 10.1: STATION CONTROL PROGRAM	94
FIGURE 10.2: USER DEFINED DIMENSIONS.....	95
FIGURE 10.3: NEW FACTORY FLOOR WITHOUT ANY OBSTACLES	95
FIGURE 10.4: OUTLINE OF THE OBSTACLE IN THE ROOM ADDED	96
FIGURE 10.5: SAVING FACTORY FLOOR MAP FOR FUTURE USE.....	96
FIGURE 10.6: OPENING A MAP FOR EDITING	97
FIGURE 10.7: FILLING THE INSIDE OF THE OBSTACLE.....	98
FIGURE 10.8: FINISHED MAP WITH GROWN OBSTACLE AND WALLS.....	98
FIGURE 10.9: LOAD MAP FOR PATH-PLANING	99
FIGURE 10.10: PATH CREATED BY MATLAB.....	100
FIGURE 10.11: PATH PLANNED USING THE QUICK PATH-PLANNING PROGRAM	101
FIGURE 10.12: SEND SELECTED FILE TO MOBROB	102
FIGURE 10.13: THE USER IS REMINDED TO SWITCH THE RF COMMUNICATION HARDWARE BACK TO RECEIVING MODE	102
FIGURE 10.14: PATH ON THE ONBOARD COMPUTER.....	103
FIGURE 10.15: MOBROB DRIVING TO ITS DESTINATION	103
FIGURE 10.16: NEW CURRENT.POS FILE	103

LIST OF TABLES

TABLE 1.1: AGVs COMPARED WITH MOBILE ROBOTS (NEL, 1997).....	2
TABLE 4.1: PIN SETTINGS FOR THE RADIOMETRIX TRANSCEIVERS.....	58
TABLE 4.2: CONTROLLING THE SENDING/RECEIVING MODES OF THE TRANSCEIVER	58
TABLE 4.3: PROCEDURES AND FUNCTIONS IN THE RF.TPU UNIT	60
TABLE 4.4: SUMMARY OF THE PROTOCOL USED IN DATA SENDING AND RECEIVING	61
TABLE 8.1: A LIST OF THE DELPHI PROCEDURES WITH A DESCRIPTION OF EACH	80
TABLE 8.2: A LIST OF THE MATLAB PROCEDURES WITH A DESCRIPTION.....	81
TABLE 8.3: A LIST OF THE PASCAL PROCEDURES WITH A DESCRIPTION	85
TABLE 8.4: SCALE USED TO DETERMINE THE FACING DIRECTION OF MOBROB	88

GLOSSARY

The following acronyms are used in this document.

Acronym	Description
A/D	Analogue to Digital conversion
AC	Alternating Current
AGV	Automated Guided Vehicle
CAD	Computer Aided Design
CAD/CAM	Computer Aided Design and Manufacturing
CAE	Computer Aided Engineering
CAM	Computer Aided Manufacturing
CASA/SME	Computer and Automated Systems Association of the Society of Manufacturing Engineers
CIM	Computer Integrated Manufacturing
COM1	Serial Communication Port
D/A	Digital to Analogue conversion
DC	Direct Current
DGPS	Differential Global Positioning System
GPS	Global Positioning System
Hz	Hertz

I/O	Input/Output
JIT	Just-in-time
LED	Light Emitting Diode
MOBROB	Mobile Robot
MRP	Materials Requirement Planning
NC	Numerical Control
PC	Personal Computer
PCMCIA	Personal Computer Manufacturers Communications Interface Adapter
RF	Radio Frequency
SA	Selective Availability
SDP	Shortest Distance Path
SENROB	Centre for Robotics at the University of Stellenbosch
SME	Society of Manufacturing Engineers
US	United States
VGRAPH	Visibility Graph

SECTION ONE: INTRODUCTION

This section gives a short introduction to this project. The problem is stated, objectives set, limitations given and the rest of the document is structured.

1 INTRODUCTION

One of the research programs currently active at the Department of Industrial Engineering (University of Stellenbosch) is the development of a mobile robot. In the rest of this document, this robot will be referred to as *MOBROB*, which is an acronym for the words *MOBile ROB*ot.

This project was started in 1993 with the development of fuzzy logic control software for an automated guided vehicle by Deist (1993). Although an automated guided vehicle (AGV) and a mobile robot are not two different names for the same object, they are closely related and Deist's thesis was the start of the MOBROB project. Nel (1997) summarised the differences between AGV's and mobile robots as given in the following table.

TABLE 1.1: AGVs COMPARED WITH MOBILE ROBOTS (NEL, 1997)

	AGV	Mobile Robot
Size	Very Small to Very Big	Medium
Guidance	Fixed paths	Free-ranging
Speed	Fast	Slow
Accuracy	Very Good	Acceptable
Obstacle Avoidance	Not Possible	Possible
Power Source	External	On board

The next step in the MOBROB project was to design and build the mobile robot. Nel completed this part in 1997. The objectives of his project were to:

- Study the basic concepts of the design of a mobile robot, mobile robot positioning systems, path-planning and fuzzy logic controllers.
- Design and construct a mobile robot.

- Find suitable suppliers for the required electronic components and to integrate the shipping and assembly.
- Implement fuzzy logic control software previously developed by Deist (1993) on the MOBROB project.
- Integrate the control, path-planning and navigational modules to enable MOBROB to be steered from a start position to a goal position in an obstacle-filled environment.

From these objectives it should be obvious that this project may be called the heart of the MOBROB project.

The other research projects done on the overall MOBROB project were all final year projects and they are as follows:

- The Mapping and Path-planning of an Industrial Mobile Robot (Van Rooyen, 1997),
- A Study of the Different Navigational Techniques for a Mobile Robot (Mentz, 1997), and
- Communication with a Mobile Robot using Radio Frequencies (Steenkamp, 1998).

These projects are all very important for the proper functioning of the robot and not one of them can function without the other.

1.1 PROBLEM STATEMENT

The problem arose when these subsets of the MOBROB project (or research projects done on the overall MOBROB project) had to be integrated.

Currently, there are five subsets: Two master theses and the three final year projects as mentioned above. These subsets appear to be unfriendly to integration, owing to the fact that different people completed the projects without taking integration into account. Another possible cause for this incompatibility may be the fact that three of the five projects were done simultaneously in 1997. It was therefore difficult to know the outcome (such as format needed for input parameter as well as the output parameters of each subset) of the projects.

All the subsets must be altered before the integration of the project is possible. The integration of the project is necessary before the first developmental stage of the project can be concluded and the second developmental stage of the project can be started. The second stage of the project will include aspects like the

docking of the robot, adding a working arm to the robot and scheduling the robot. This broader perspective of the MOBROB project is illustrated in Figure 1.1.

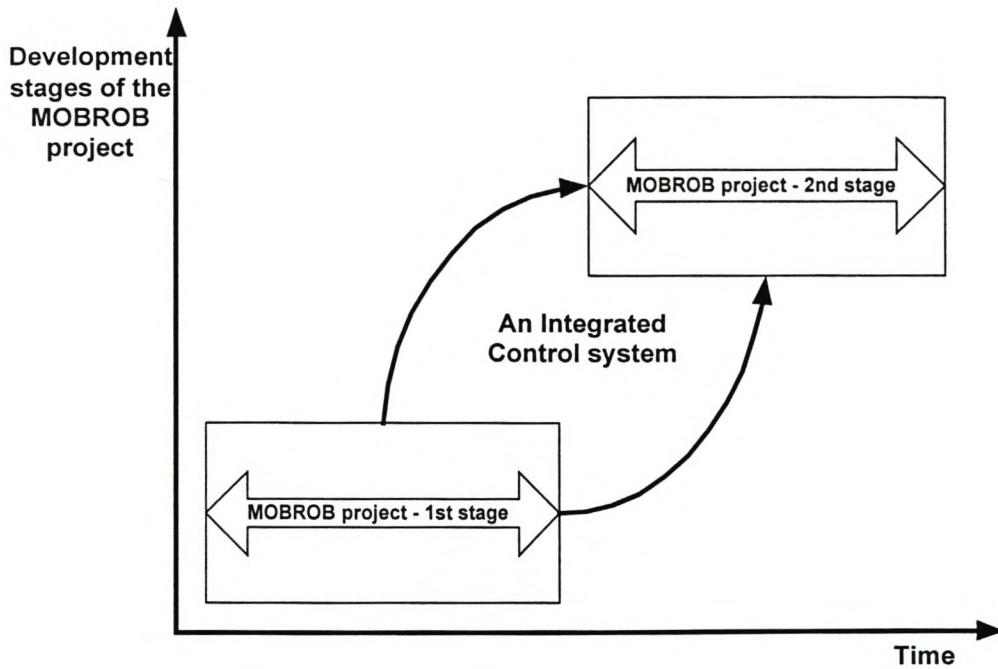


FIGURE 1.1: THESIS PROBLEM PLACED IN PERSPECTIVE (WITH THE MOBROB PROJECT)

1.2 THESIS OBJECTIVES

The thesis forms part of the MOBROB project and the objectives of the thesis are to:

- Do a study on CIM systems.
- Study the basic concepts of Mobile robotics.
- Get acquainted with the MOBROB project and all its subsets.
- Design, construct (in an applicable way), test, and implement (on the MOBROB project) an integrated control system for a mobile robot.

If all the above-mentioned objectives were reached, the product delivered through this thesis would be:

An integrated control system for

- a) The control of a mobile robot,
- b) The control of the local environment of the mobile robot, and
- c) The control of the global environment of the mobile robot.

1.3 LIMITATIONS

The author did not experience any real limitations while completing this project.

The lack of knowledge is not seen as a limitation, but rather as a challenge, whereas hardware breakage is seen as a time-wasting occurrence.

1.4 STRUCTURE OF THE THESIS

The thesis can be divided into four different sections. This is illustrated in Figure 1.2.

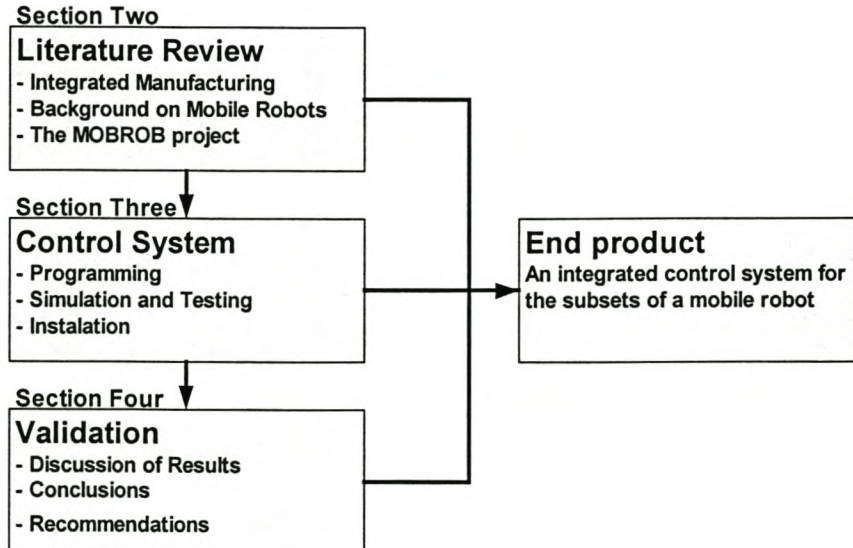


FIGURE 1.2: DIFFERENT SECTIONS OF THIS DOCUMENT

The first section of this thesis is the introduction (chapter one).

The second section of the thesis (chapters two, three, four and five) consists of a literature review. This review is about computer-integrated manufacturing (CIM) (chapter two), mobile robots in general (chapter three), the MOBROB project in the AS-IS State (chapter four) and conclusions (chapter five).

Chapters six, seven, eight and nine are devoted to an explanation of the integrated control system and its components. These chapters form the third section of the thesis that the author refers to as the design of the integrated control system.

The fourth section of the thesis is the validation and finishing of the project, where results are discussed (chapter ten), the project summarised (chapter eleven), and conclusions (chapter twelve) and recommendations (chapter thirteen) are made.

All guides, program codes, bulky results, mathematical explanations, etc. are placed in the appendixes at the end of the report.

SECTION TWO: LITERATURE REVIEW

This section gives a thorough literature review of the literature studied to complete this project.

2 COMPUTER INTEGRATED MANUFACTURING (CIM)

This chapter discusses Computer Integrated Manufacturing (CIM) very briefly.

2.1 INTRODUCTION

According to Singh (1996), design and manufacturing are the core activities for realising a marketable and profitable product. It is also true that these fields have changed dramatically over the last few decades. The development of technologies (such as CAD/CAM, CNC, CIM, Artificial Intelligence, AGV's, Robotics, Mobile robotics, PLC's, Flexible Manufacturing Systems and Flexible Manufacturing Cells) resulted in radical changes in the design and manufacturing world, as it was known 30 years ago.

Dr J Harrington, Jr, coined the name computer-integrated manufacturing (CIM) in 1973. An integrated approach to the enterprise that lead to localised optimisation was required, instead of highly-fragmented manufacturing operations. Considering the current and future market trends for customised products, the formation of virtual organisations is considered a strategic weapon to combat competition and stay in business over the long term. For virtual organisations to succeed in achieving corporate goals and objectives, as well as for its customers and suppliers, there is a greater need for integrated solutions to the many problems across the enterprise.

Ránky (1986) defined CIM as follows:

"Computer Integrated Manufacturing (CIM) is concerned with providing computer assistance, control and high level integrated automation at all levels of the manufacturing industries, by linking islands of automation into a distributed processing system. The technology used in CIM makes intensive use of distributed computer networks and data processing techniques, Artificial Intelligence and Data Base Management Systems (Figure 2.1).

In the more recent years a paradigm shift has occurred in the definition of CIM. Previously, the concept of CIM activities was limited to manufacturing operations. However, mass customisation of products needs much more than manufacturing integration. It demands the creation of virtual organisations to undertake specific projects or market niches.

In his book on CIM, Singh (1996) gives the following definition for CIM:

"CIM is the integration of the total manufacturing enterprise through the use of integrated systems and data communications coupled with new managerial philosophies that prove organisational and personnel efficiency."

-The Computer and Automation Systems Association of the Society of Manufacturing Engineers

In another book on CIM, Vajpayee (1995) gives several definitions for CIM. Some of these definitions follow:

"CIM is not applying computers to the design of the products of the company. That is computer aided design (CAD)! It is not using them as tools for part and assembly analysis. That is computer aided engineering (CAE)! It is not using computers to aid in the development of part programs to drive machine tools. That is computer aided manufacturing (CAM)! It is not materials requirements planning (MRP) or just-in-time (JIT) or any other method for developing the production schedule. It is not automated identification, data collection, or data acquisition. It is not simulation or modelling of any materials handling or robots or anything else like that. Taken by themselves, they are the application of computer technology to the process of manufacturing. But taken by themselves they only create the islands of automation."

-Leo Roth Klein, Manufacturing Control Systems, Inc.

*"Computer integrated manufacturing is a broad term covering all technologies and **soft automation** used to manage the resources for cost-effective production of tangible goods."*

-S. Kant Vajpayee, University of Southern Mississippi

"CIM is an opportunity for realigning your two most fundamental resources: people and technology. CIM is a lot more than the integration of mechanical, electrical, and even information systems. It's an understanding of the new way to manage."

-Charles Savage, president, Savage Associates

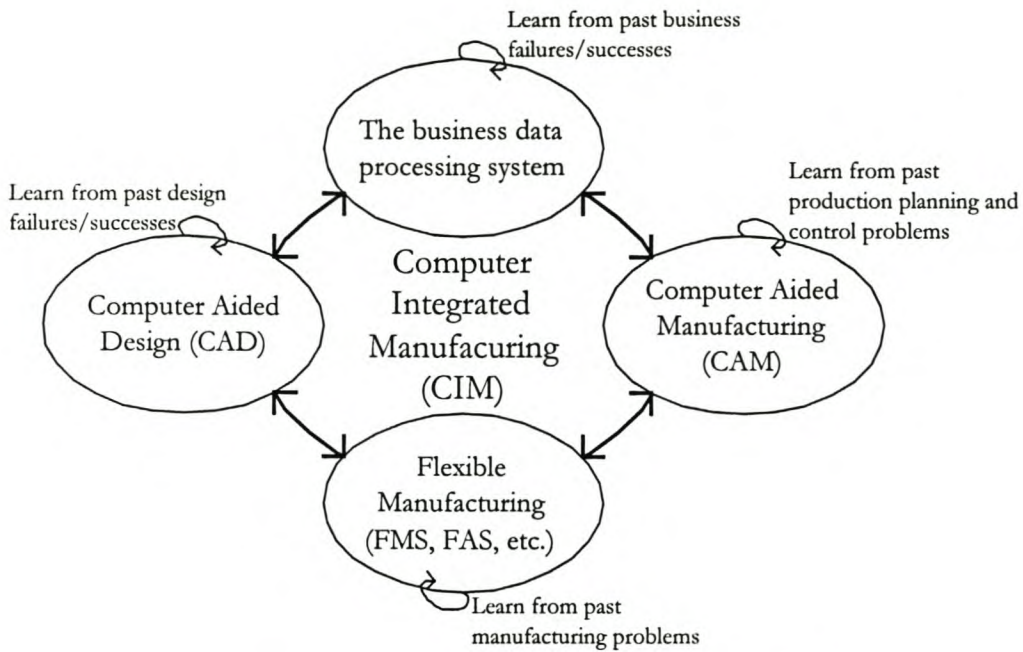


FIGURE 2.1: A CIM SYSTEM CONCEPT (RÁNKY (1986))

It is clear that these definitions cover a broad variety of topics and that CIM is not only the integration of manufacturing processes, but includes all the different aspects of a manufacturing enterprise such as scheduling, material selection, quality assurance, design processes, ergonomics, flexible manufacturing, decision theory, and management science. However, the emphasis is still, and maybe even more than in the past, on integration. Owing to the new demands in the manufacturing world it is of utmost importance for a manufacturing enterprise to integrate all the aspects of the enterprise.

2.2 SKILLS REQUIRED FOR THE SUCCESSFUL IMPLEMENTATION OF A CIM SYSTEM

From the introduction above and definitions of CIM it is clear that the “CIM engineer” needs advanced engineering and communicative skills. More than fifteen years ago, Ránky (1986) identified the skills a “CIM engineer” needs for the successful implementation of CIM systems. Some of these skills are still necessary today and are therefore included in this document.

The CIM engineer must have:

- ☞ The ability to describe, define and analyse different computer-integrated models as a system, by

specifying its components and their functional relationships,

- ☞ Knowledge of computers and interfaces used in intelligent machines,
- ☞ The ability to develop several different logical structures for the same specified problem.
- ☞ The ability to outline, design and implement medium and large size software projects in the areas of database processing, computer graphics, real-time control, digital network communication and application software design,
- ☞ The necessary thoroughness and accuracy in his/her work to be able to overcome the occasional panic situations and complete the vast amount of work regarding information processing.
- ☞ The necessary people skills to co-operate with other people working on the system.

2.3 COMPLEXITY OF SYSTEMS

The concept of a 'system' in the mathematical sense applies to any process, whether physical, economic or social. Thus, any company or even its component parts (factory, workshop, etc.) may be considered as a system.

The complexity of a system may be modelled by the variety of the various states, which it is capable of adopting. In Figure 2.2 system S has three components C1, C2, C3 capable of being in a , b and c states respectively. The variety V of the system is abc or a^3 if $a = b = c$. When several systems are combined together, their varieties or complexities do not add, but multiply (Figure 2.3).

Controlling a system involves associating with another system whose role is to keep the variety of the results or objectives as small as possible.

The variety of results or objectives (V_0) cannot be less than

$$V_0 = \frac{V}{V_c} = \frac{\text{variety of systems to be managed}}{\text{variety of control system}} \quad \text{Equation 1}$$

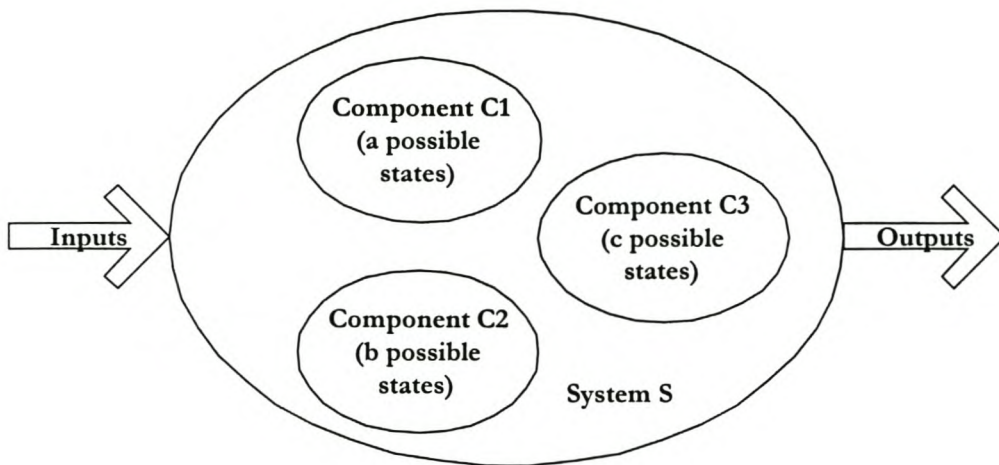


FIGURE 2.2: THE VARIETY OF A SYSTEM (WALDNER (1990))

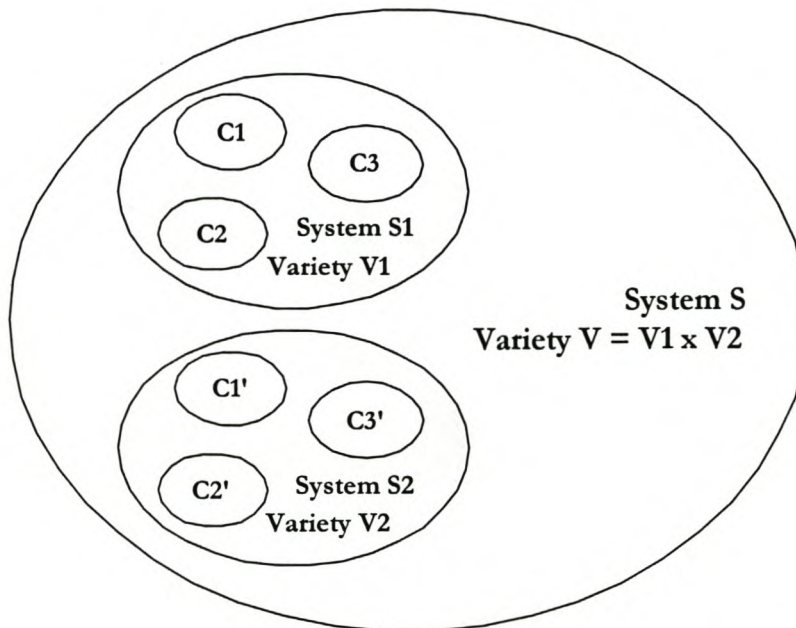


FIGURE 2.3: THE VARIETY OF A COMBINATION OF SYSTEMS (WALDNER (1990))

Since the variety of the results is a minimum, it can only decrease if the variety of the control system increases.

This expresses the *law of requisite variety*, which establishes that only the variety of the control system can reduce that resulting from the process to be controlled, and that only complexity can destroy complexity.

This fundamental principle shows that the regulation of a system in the strict sense demands a control system whose complexity is equal to, or greater than, that of the system to be managed.

In view of the prohibitive number of dimensions, which such a system would take on, it is easy to appreciate the importance of the procedure for simplifying the functions of the company before any attempt at their integration, even when using facilities as powerful as the largest computers. This step is an essential prerequisite for the success of any CIM project.

2.4 CIM WHEEL

CASA/SME has suggested a framework, the CIM wheel, to elucidate the meaning of CIM. Formed by SME in 1975, CASA is an interest group of manufacturing professionals. The CIM wheel, developed by CASA/SME's Technical Council, is shown in Figure 2.4. It provides a clear portrayal of the relationships among all parts of an enterprise. It depicts a central core (integrated systems architecture) that handles the common manufacturing data and is concerned with information resources management and communications. The radial sectors surrounding the core (wheel hub) represent the various activities of manufacturing, such as design, material processing, and inspection. These activities have been grouped into three categories – manufacturing planning and control, product/process, and factory automation – as depicted in the wheel's inner rim. The outer rim represents the upper management functions, grouped into four categories: strategic planning, marketing, manufacturing and human resource management, and finance. The original wheel did not have the outer rim. This rim was added in 1985 to emphasise the need of including both management and technology functions within the scope of CIM. As the wheel illustrates, CIM is broad enough to encompass all aspects of the manufacturing enterprise and its management, including those of personnel and finance.

2.5 BENEFITS OF CIM

Similar to the variations in CIM definitions, the benefits to companies depend on their experience with CIM. In general CIM benefits can be grouped into tangible and intangible categories. Vajpayee (1995) gives a few benefits for CIM divided into these categories in his book on the topic.

1) Tangible Benefits

- Higher profits,
- Less direct labour,
- Increased machine use,

- Reduced scrap and rework,
- Increased factory capacity,
- Reduced inventory,
- Shortened new product development time,
- Fewer missed delivery dates, and
- Decreased warranty costs.

2) Intangible Benefits

- Higher employee morale,
- Safer working environment,
- Improved customer image,
- Greater scheduling flexibility,
- Greater ease in recruiting new employees,
- Increased job security, and
- More opportunity for upgrading skills.

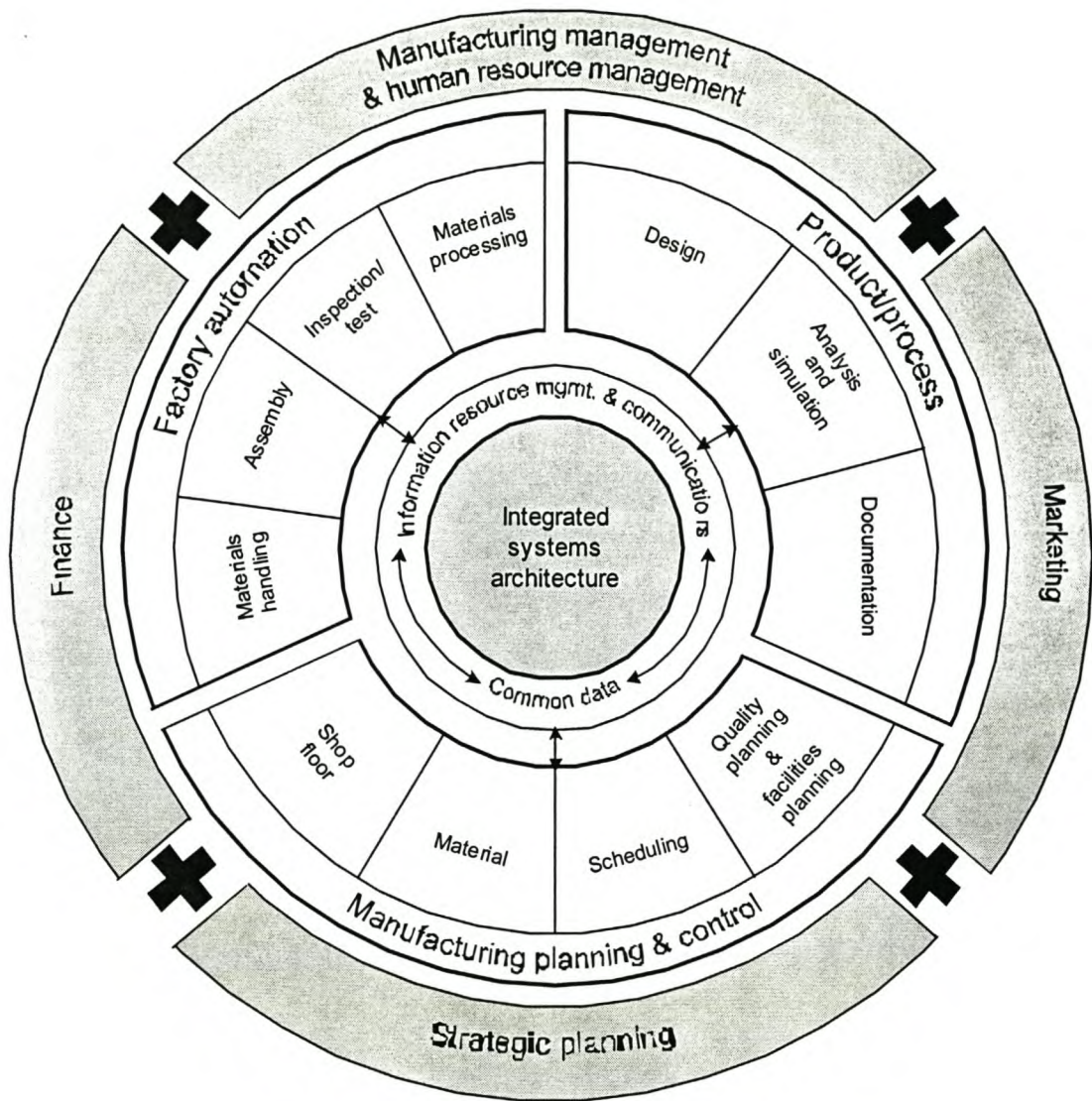


FIGURE 2.4: THE SME CIM WHEEL © 1985, SOCIETY OF MANUFACTURING ENGINEERS, DEARBORN, MI
48121

2.6 TRENDS CONCERNING CIM

There are a number of worldwide trends that are the direct result of CIM.

- Interest in CIM is worldwide, with many people calling it a do-or-die phenomenon. World-class manufacturers have in the past and are still adopting CIM.

- CIM solutions are usually not generic; they are customised for a particular company. There is, however, a trend to package CIM solutions in a generic way.
- CIM's impact on manufacturing is in essence evolutionary; but its impact on discrete manufacturing is revolutionary.
- CIM is likely to do to manufacturing what mechanisation did to farming, with fewer workers producing in fewer factories all the goods needed by society at an affordable price.

2.7 SUMMARY

This chapter introduced the concept and history of Computer Integrated Manufacturing or CIM. Simply stated, CIM involves the integrated application of computers in manufacturing. CIM is an umbrella term, covering all manufacturing functions and activities, both technical and managerial.

Computers were first used in manufacturing for business functions and inventory control. Next computers were applied to design under CAD, followed by CAM, and then CAD/CAM. By 1975, CIM became a concept, and the 1980s saw it expand into a technology until it became one of the most talked-about topics in manufacturing.

The reason why CIM was surveyed is that many of the principles of CIM apply to the integration of the subsets of MOBROB.

3 LITERATURE REVIEW ON MOBILE ROBOTS

A mobile robot must pass the following test (Salichs and Moreno 2000): *‘The robot is placed in an environment that is unknown, large, complex and dynamic. After a time needed by the robot to explore the environment, the robot must be able to go to any selected place, trying to minimise a cost function (e.g. time, energy, etc.).* Certain questions are addressed in this work, concerning motion control, mapping or modelling of the environment and path-planning. Answers given to these questions in the past are then tested against this test. It was found that this test remains unsolved.

In this chapter mobile robotics are discussed in general, addressing design, positioning, mapping, path-planning, communication and docking techniques.

3.1 MOBILE ROBOT DESIGN

Various different aspects are important when doing mobile robot design. The most important aspects are now discussed briefly.

3.1.1 STEERING ARRANGEMENTS

The single feature that determines the behaviour of a mobile robot is the way in which it is steered. The steering arrangement also affects the total cost of the vehicle, because driving motors are usually expensive compared to the other components of a mobile robot. The most commonly used steering arrangements will now be discussed.

Differential steering arrangement: In a differential steering arrangement two wheels (fitted on a common axis), are driven independently. By controlling the speed of each wheel, it enables the vehicle to drive straight, to turn in place, or to move in an arc. To balance the vehicle, one or more castors form part of this steering arrangement (Figure 3.1).

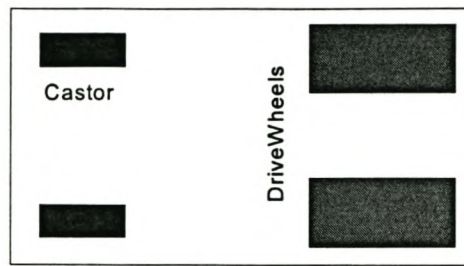


FIGURE 3.1: DIFFERENTIAL STEERING ARRANGEMENT

The castors can be placed on both sides of the driving wheels, or just on one side. The latter is preferable because of increased stability and reduced wheel slippage. Wheel slippage could be caused by castors on both sides of the driven wheel, as illustrated in Figure 3.2.

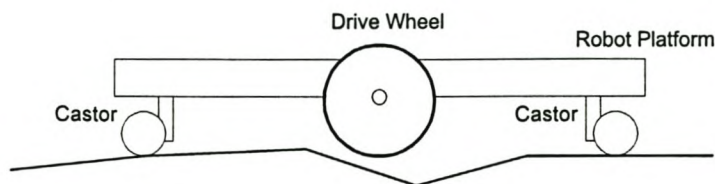


FIGURE 3.2: CASTORS CAUSING WHEEL SLIPPAGE

- **Car and tricycle-type steering arrangement:** A car arrangement as shown in Figure 3.3 consists of four wheels, of which two are driven and two are used for steering. The tricycle type works on the same principle, but because of its single steering wheel, it is mechanically simpler. In rough terrain, the steering wheels (for car and tricycle type) can also be driven, but this adds complexity and cost.

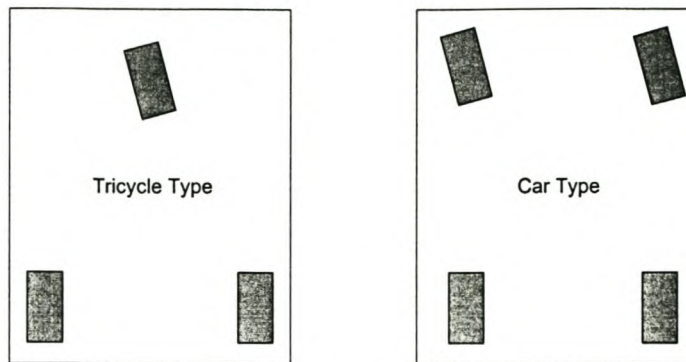


FIGURE 3.3: CAR AND TRICYCLE TYPE DRIVES

- **Synchro drive:** In a synchro steering arrangement (Figure 3.4), all wheels (usually three) are driven and are used for steering. All the wheels always point in the same direction. Steering all the wheels simultaneously changes the direction of the vehicle. The wheels thus change direction, but the chassis continuously points in the same direction.

Mobile robot navigation normally requires sensors to be placed on the front side of the vehicle. With synchro drive, it is not possible to define a front side of the robot because the direction of the robot never changes. It is therefore necessary to fix sensors around the whole perimeter of the robot, thereby increasing cost and complexity. Because the robot does not have a fixed front side, it could be recommended that the shape of a synchro drive robot should always be round.

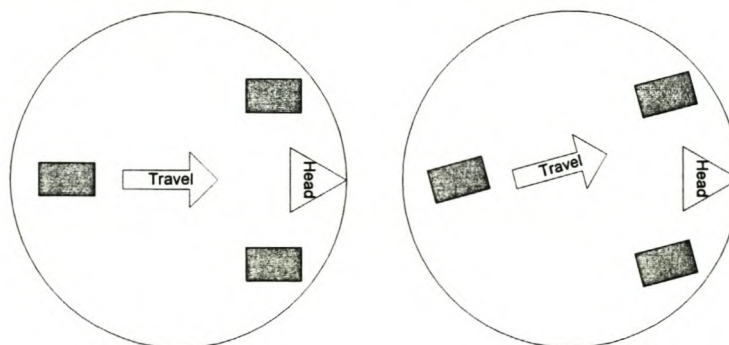


FIGURE 3.4: SYNCHRO DRIVE

3.1.2 SHAPE AND SIZE

The most common mobile robot shapes are circular, square and rectangular. The shape of the robot depends on its functions and payload. A general rule of thumb states that it is always easier to control round-shaped robots, and they also have better obstacle-avoiding abilities. This is because a round shaped robot has the ability to rotate round its centre axis, while very close to obstacles. The size of the robot is only dependent on the payload, and the components used in the vehicle.

3.1.3 POWER SOURCE

A mobile robot can be driven by several different power sources. In some research projects done on mobile robots, ordinary vehicles, for example forklifts and four-wheeled motorcycles were used. Thus, the power supply for a mobile robot can basically be any available type of power source. The only limitation is that it has to be fitted on board the vehicle.

3.1.4 DRIVING SOURCE

In every mobile robot a driving source is needed to propel the vehicle. The most commonly used driving sources are electrical motors and petrol/diesel engines. These driving sources are normally used in combination with gear reduction systems. Electrical motors can either be AC, DC or stepper motors. Mobile robots developed for laboratory environments normally have DC or stepper motors. Stepper motors have the advantage that they can be accurately controlled without encoder feedback. DC motors need encoder feedback for accurate control, but are more suitable for applications where heavy loads have to be moved.

3.1.5 CONTROL COMPUTER

Several forms of control computers are available for use on mobile robots. One of the things that must be kept in mind when selecting a computer is that some kind of software/hardware interface is needed between the computer source code and the components, which receive input or give output to the computer. Four possibilities are identified.

- **Notebook computer:** A notebook computer is lightweight, battery-driven and compact, and therefore very suitable for mobile robot applications. Notebooks have two different ways of receiving/sending input/output (I/O). The first is PCMCIA slots, which are used to host credit card-size computer cards. Readily available cards can perform Analogue to Digital (A/D) and Digital to Analogue (D/A) conversions, and have several I/O lines.

- **Notebook computer with docking station:** Another option is to use the notebook in combination with a docking station. The docking station provides ISA slots (available in every PC), which can be used to host full size, or half size computer cards. An advantage of this system is that it can host several cards at the same time, and the full size cards usually have more functions (I/O, A/D and D/A) than the PCMCIA cards. A disadvantage of a notebook-docking station system is that it requires 220V AC to operate. 220V AC is not readily available on a battery-powered vehicle, but can be obtained by using a DC to AC inverter.
- **Battery operable PC:** A normal personal computer can be used on battery power, by using a special power supply. The power supply fits into the same space as the normal 220V AC power supply, and it can operate from a wide range of DC inputs. However, the power supply can only be used for the computer, and cannot supply power to the monitor. This problem can be overcome by using a battery-operated, flat panel computer display; similar to the types used in a notebook. The advantage of such a system is that it does not require 220V AC and that the computer can host more ISA cards than a notebook-docking station combination. The only disadvantage is the high cost of the flat panel display.
- **Industrial type computers:** These computers are very rugged, multi-slotted, portable computers, specially designed for industrial use. They can operate on battery power, and can host a fair amount of full-size computer cards. They are ideally suited for use in a mobile robot, but are more expensive than a notebook-docking station-inverter combination.

3.1.6 SOFTWARE-HARDWARE INTERFACE

A software-hardware interface enables the computer to communicate with the different components. This communication can be done via the parallel ports, serial ports and the ISA slots. Communication via the parallel and serial ports is relatively easy and cheap, but is limited to the amount of I/O lines. A large variety of computer cards for the control of a mobile robot, is available. Two types of these cards are more suitable for mobile robots, namely motion control cards and analogue and digital I/O cards.

- **Motion control cards:** Motion control cards are normally used in industry to control NC machines. They can be programmed by G-codes to perform different machining tasks. This type of card can be used in a mobile robot to control the motors.
- **Analogue and digital I/O cards:** These cards can be used to receive all relevant information regarding the control of a mobile robot, for example the sensor readings or input from touch-sensitive bumpers. The cards can also be used to transmit signals that can be used, for example,

to control the motors or navigational system.

3.2 MOBILE ROBOT POSITIONING

A position system for a mobile robot normally consists of a two dimensional space (x and y co-ordinates), but it is also necessary to know the orientation angle Φ of the robot. Borenstein, et al. (1995) defined seven categories for positioning systems that can be grouped into two main groups, namely:

Relative position measurements (dead reckoning):

- Odometry
- Inertial navigation

Absolute positioning measurement (reference-based systems):

- Magnetic compasses
- Active beacons
- Global positioning systems
- Landmark navigation
- Map-based positioning

These are not the only methods used for positioning. Hoppenot et al (2000) describes a method using inputs from ultrasonic sensors to determine position. The robot must have full a priori knowledge (see paragraph 3.3.1 for a definition) of the environment available for this method. These categories will be reviewed in the following few paragraphs.

Meng, Sun & Cao (2000) presents another method. This method use an adaptive extended Kalman filter-based system with sonars to determine the position of the robot in an indoor structured environment.

3.2.1 ODOMETRY

Odometry is based on the assumption that wheel revolutions can be translated into linear displacement relative to the floor. This hypothesis is only of limited validity. One excessive example is wheel slippage: if

one wheel were to slip on an oil spill, then the connected encoder would register wheel revolutions, even though these revolutions would not be consistent to the linear displacement of the wheel. Even though this is an extreme example, there are several other more subtle reasons for inaccuracies in the translation of wheel encoder readings into linear motion. All these errors can be categorised into two main categories.

Systematic errors

- Unequal wheel diameters
- Average of both wheel diameters differ from nominal diameter
- Misalignment of wheels
- Uncertainty about the effective wheelbase (due to non-point wheel contact with the floor)
- Limited encoder resolution
- Limited encoder resolution sampling rate

Non-systematic errors

- Travel over uneven floor
- Travel over unexpected objects on the floor
- Wheel slippage due to
- Slippery floors
- Over-acceleration
- Fast turning (skidding)
- External forces (interaction with external bodies)
- Internal forces (castor wheels)
- Non-point wheel contact with the floor

Borenstein, et al. (1995) Introduced “UMBmark,” a method for measuring and correcting systematic odometry errors in differential-drive mobile robots. With this method they were able to reduce the *systematic*

odometry error of an uncalibrated robot by one order of magnitude.

A method for detecting and rejecting *non-systematic* odometry errors in mobile robots was also developed. With this method, two collaborating platforms continuously and mutually correct their non-systematic (and certain systematic) odometry errors, even while both platforms are in motion. Linear displacement sensors connect the platforms to each other; thus the positional error of one can be corrected by knowing the correct position of the other.

3.2.2 INERTIAL NAVIGATION

Inertial navigation uses gyroscopes and accelerometers to measure the rate of rotation and acceleration, respectively. The measurements from the accelerometers are integrated twice to yield position, while the measurements from gyroscopes are used to determine the orientation of the vehicle. An advantage of inertial navigation systems is that they are self-contained; that is, they do not need external references. A disadvantage is that the sensor data may drift from time to time, leading to large positional errors. Accelerometers are sensitive to uneven surfaces, because disturbance from a perfectly horizontal position will cause the sensor to detect a component of the gravitational acceleration. Inertial sensors are thus mostly suitable for accurate positioning over an extended period.

Gyroscopes are of particular importance to mobile robot positioning, because as mentioned in paragraph 3.1, a small error in orientation can cause a large positional error. Odometry will thus benefit greatly if orientation errors could be detected and corrected immediately.

3.2.3 MAGNETIC COMPASSES

Vehicle heading is the most significant of the navigation parameters (x , y , and θ) in terms of its influence on accumulated dead-reckoning errors. For this reason, sensors that provide a measure of absolute heading are extremely important in solving the navigation needs of autonomous platforms. The magnetic compass is such a sensor. One disadvantage of any magnetic compass, however, is that the earth's magnetic field is often distorted near power lines or steel structures. This makes the straightforward use of geomagnetic sensors difficult for indoor applications.

Based on a variety of physical effects related to the earth's magnetic field, different sensor systems are available:

- Mechanical magnetic compasses.
- Fluxgate compasses.
- Hall-effect compasses.
- Magnetoresistive compasses.
- Magnetoelastic compasses.

The compass best suited for use with mobile robot applications is the fluxgate compass. When maintained in a level attitude, the fluxgate compass will measure the horizontal component of the earth's magnetic field, with the decided advantages of low power consumption, no moving parts, intolerance to shock and vibration, rapid start-up, and relatively low cost. If the vehicle is expected to operate over uneven terrain, the sensor coil should be gimbal-mounted and mechanically dampened to prevent serious errors introduced by the vertical component of the geomagnetic field.

3.2.4 *ACTIVE BEACONS*

Active beacon navigation systems are the most common navigation aids on ships and aeroplanes, as well as on commercial mobile robot systems. Active beacons can be detected reliably and provide accurate positioning information with minimal processing. As a result, this approach allows high sampling rates and yields high reliability, but it does also incur high cost in installation and maintenance. Accurate mounting of beacons is required for accurate positioning. Two different types of active beacon systems can be distinguished: trilateration and triangulation.

- **Trilateration** is the determination of a vehicle's position based on distance measurements to known beacon sources. In trilateration navigation systems there are usually three or more transmitters mounted at known locations in the environment and one receiver on board the robot. Conversely, there may be one transmitter on board and receivers are mounted on the walls. Using time of flight information, the system computes the distance between the stationary transmitters and the onboard receiver. Global Positioning Systems (GPS), discussed in later paragraphs of this chapter are examples of trilateration.
- **Triangulation** requires three or more active transmitters placed at known locations. A rotating sensor on board the vehicle registers the angles of three beacons relative to the longitudinal axis of the vehicle. From these measurements, the unknown x and y co-ordinates and the unknown

vehicle orientation Φ can be calculated.

3.2.5 GLOBAL POSITIONING SYSTEM

The global positioning System (GPS) was developed for military use. The system consists of 24 satellites that transmit encoded Radio Frequency (RF) signals. The exact distances from a ground receiver to three satellites are measured, and then trilateration is used to calculate the latitude, longitude and altitude of the receiver. To measure the exact distance to a satellite, it is necessary to know the exact position of the satellite, at that specific time. The United States' government deliberately applies small errors in timing and satellite position to prevent other countries using GPS against them in war. This causes an intended positional error of approximately 100m. The small errors in timing and satellite positions can be switched on and off as desired by the US government (Borenstein et al. 1996). This feature is called selective availability (SA).

The positional errors caused by selective availability can effectively be eliminated by a practice known as differential GPS (DGPS). If two GPS receivers are relatively close to each other (less than 10 km), they will experience the same error effects when viewing the same reference satellites. If one receiver is fixed at a known position, it can compute the error vector in that specific region. The error vector can thus be used to correct the positional error of the other receiver. Typically, the accuracy of a DGPS is in the region of four to six metres.

It was concluded that GPS is most suited for outdoor use. One problem with GPS in mobile robot navigation could be the absence of RF signals of three different satellites as needed for trilateration. Another problem is that the positional accuracy of GPS is inadequate for a primary (stand-alone) navigation system.

3.2.6 LANDMARK NAVIGATION

Landmarks are different characteristics that a robot can recognise from its sensory input. Landmarks can be geometric shapes (e.g., rectangles, lines, and circles), and they may include extra information (e.g., in the form of bar codes). In general, landmarks have a fixed and known position, comparative to which a robot can locate itself. Landmarks are chosen with care, so that they are easy to identify. Before a robot can use landmarks for navigation, the characteristics of the landmarks must be known and stored in the robot's memory. The main task in localisation is then to recognise the landmark's reliably and to calculate the robot's position (Connessons & Vasiljevic, 2000).

In order to simplify the problem of landmark acquisition it is often assumed that the current robot position

and orientation are known approximately, so that the robot only needs to look for landmarks in a limited area. For this reason good odometry accuracy is a prerequisite for successful landmark detection.

Some approaches fall between landmark and map-based positioning (see Paragraph 2.7). Sensors are used to sense the environment and then extract distinct structures that serve as landmarks for navigation in the future.

Borenstein et al (1996) addresses two types of landmarks: “artificial” and “natural” landmarks. They define the terms “natural landmarks” and “artificial landmarks” as follows: natural landmarks are those objects or features that are already in the environment and have a function other than robot navigation; artificial landmarks are specially designed objects or markers that need to be placed in the environment with the sole purpose of enabling robot navigation.

- **Natural landmarks** are most suitable in highly-structured environments such as corridors, manufacturing floors, and hospitals. The best natural landmarks are usually man-made. The main problem in natural landmark navigation is to detect and match characteristic features from sensory inputs. The sensor of choice for this task is computer vision. Most computer vision-based natural landmarks are long vertical edges, such as doors, wall junctions, and ceiling lights.
- **Artificial landmarks** are specially designed objects or markers that need to be placed in the environment with the sole purpose of enabling robot navigation. Artificial landmarks are specially designed to yield a high contrast with its surroundings. The robot also knows the exact size and shape in advance. It is thus easier for the robot to detect these landmarks.

A few characteristics of landmark-based navigation can be summarised as follows:

- Natural landmarks offer flexibility and require no modifications to the environment.
- Artificial landmarks are inexpensive and can have additional information encoded as patterns or shapes.
- The maximal effective distance between robot and landmark is substantially shorter than in active beacon systems.
- The positioning accuracy depends on the distance and angle between the robot and the landmark. Landmark navigation is rather inaccurate when the robot is further away from the landmark. A higher degree of accuracy is obtained only when the robot is near a landmark.
- Substantially more processing is necessary than with active beacon systems. In many cases onboard computers cannot process natural landmark algorithms quickly enough for real-time

motion.

- Ambient conditions, such as lighting, can be problematic; in marginal visibility landmarks may not be recognised at all or other objects in the environment with similar features can be mistaken for a legitimate landmark. This is a serious problem because it may result in a completely erroneous determination of the robot's position.
- Landmarks must be available in the work environment around the robot.
- Landmark-based navigation requires an approximate starting location so that the robot knows where to look for landmarks. If the starting position is not known, the robot has to conduct a time-consuming search process. This search process may go wrong and may yield an erroneous interpretation of the objects in the scene.
- A database of landmarks and their location in the environment must be maintained.
- There is only limited commercial support for natural landmark-based techniques.

3.2.7 *MAP-BASED POSITIONING*

Map-based positioning systems can work in two ways. The first is where the robot uses its sensors to construct a map of its surroundings. In normal operation, a comparison between the sensor input and the map is used to determine the robot's position. The second is where a pre-stored map (which could have been generated by CAD) is compared with the sensor input. In both systems, it is possible to update the map as new obstacles appear.

An advantage of a map-based positioning system is that it uses naturally occurring structures of typical indoor environments, without any modification of the environment. The speed of the system depends on the processing power, while the positional accuracy is dependent on the sensing capacity, and the accuracy of the sensor map.

3.3 MAPPING AND PATH-PLANNING

Path-planning is the task of finding a continuous and obstacle-free trajectory from the present position to the goal position, while optimising some cost function. The path-planning method used is dependent on the assumption according to which the mobile robot is controlled.

3.3.1 OPERATING WITH FULL A PRIORI INFORMATION

Operating with full a priori information about the environment is the first assumption on which a mobile robot can be controlled. A priori knowledge about the environment inevitably means that the mobile robot needs to have some world representation or world map from which it can operate, the world being the environment that the mobile robot needs to function in.

The world map has to serve as a basis for operations such as path planning, obstacle avoidance and position estimation. Complicated world representations suitable for real time control of the mobile robot, being cumbersome to update, require a great deal of available memory, and the computational/processing time is unacceptably long. The ability to update or edit a world map is essential, as it is inevitable that changes will occur in the factory.

3.3.2 OPERATING WITH NO A PRIORI KNOWLEDGE

Operating with no a priori knowledge about the environment is the second assumption on which a mobile robot can be controlled. This type of mobile robot control is based on the perception of the environment through a variety of sensors and some means of extracting meaningful information from this data.

This representation of the environment is then analysed to search for a local obstacle-free path. This method is usually associated with localisation, as compared to global path planning. Localisation, also referred to as absolute localisation, occurs when position is being determined from objects currently in the environment, rather than position taken from a co-ordinate origin. The next move the mobile robot will make, is therefore completely independent of assumptions about previous movements.

A slight difference to this type of mobile robot control is for the mobile robot to have some generic knowledge about the environment. This means that the mobile robot still does not have any a priori knowledge about the specific environment, but does have some knowledge about generic environments. It is therefore able to distinguish, e.g. a door from a wall. This is done by gathering data from its sensors and then comparing that data with a generic-model database in order to recognise some features of its environment.

3.3.3 MAPPING

Without a good and accurate representation of the environment of the mobile robot, the robot would not be very useful, because it is impossible to do path-planning without a map. A good representation and accurate

position determination can mean the difference between success and failure. Programmable systems do not have this problem, but lack in the degree of flexibility and autonomy of systems that are not programmable.

1) Real World Mapping

A real world environment has an infinite number of points in it, depending on the number of decimals with which each point is represented. The problem is that mobile robots systems use a finite number of discrete memory. Therefore, large constraints have to be placed on the environment in order to represent it effectively and to use memory efficiently on a mobile system. For more complex representation, which usually results in better performance, although it is largely dependent on the average error in positioning determination, more computations are required to analyse it. With the fast development of computers, memory per Megabyte is less expensive than some years ago, but it is still a finite number and must be used as effectively as possible.

The easiest way to accomplish the function of real world mapping, is to divide the real world into a fixed block-sized grid. The size of the block determines the resolution, and the size of the grid, the area to be covered by the map. As an obstacle is placed in the map, it is unlikely that the obstacle will fit exactly into a block and therefore, some compromises must be made.

One of the more obvious solutions is to accept that, as the system is prone to error in any case, an overcautious mapping will result in better performance on average. This approach is followed in the design of the mobile robot system. The advantage here is that the mapping is quickly updateable if obstacles are added or removed.

Another solution to the problem is to subdivide each block further into smaller blocks, only if an obstacle occupies it. This method is called cell decomposition and ensures that the whole obstacle can be accurately mapped. The advantage of this method above the one described previously, is that it is complete. That implies that it is guaranteed to have a free path between two points, where the first example may fail in this respect. It, however, requires more sophisticated search techniques to find a path. The computational complexity of this technique makes it unsuitable for use in a simple robot system.

2) Static and Dynamic Mapping

Static mapping or path-planning assumes that the mobile robot has full *a priori* knowledge of the environment. This knowledge can be gained by either programming the system with a map, or allowing the system to construct its own map by using the information gained by the robot's sensors.

Quasi-static mapping is another process that can be used to construct a map of the environment and is based on dynamic mapping techniques. Quasi-static mapping is a process where the system is placed in a completely unknown environment and commanded to traverse the distance between two points. As it has no map available to it, the solution to the problem is to take the shortest route, i.e. a straight line connecting the two points. The deviation here from pure dynamic mapping is clearly visible. The system has taken in no information about the environment to help it in its search, whereas dynamic mapping would first get a world perception before deciding on an appropriate course of action.

In an obstacle-filled environment, the system is likely to collide with obstacles as it is moving to the target position. The system maps the obstacle, as with static mapping, and then using this new map, plans a path to the target in the same way static mapping would have done. As the robot moves along, it encounters more obstacles, and is involved in a process of dynamically mapping the environment. This is a process where mapping and path-planning are done simultaneously.

Greedy mapping presented by Koenig et al (2001) is a method used to build a map from sensor readings, thus using dynamic mapping techniques. The robot will always move to the closest location that it has not visited (or observed) yet, until the terrain is mapped.

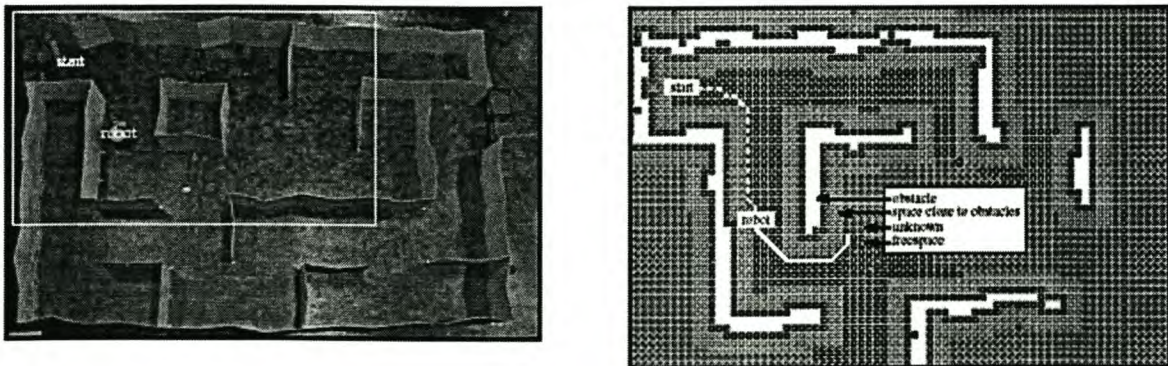


FIGURE 3.5: GREEDY MAPPING (KOENIG ET AL, 2001)

Greedy Mapping takes advantage of prior knowledge about parts of the terrain (if available) since it uses all of its knowledge about the terrain when determining which unvisited (or unobserved) location is closest to the robot and how to get there quickly. It does not matter whether this knowledge was previously acquired by the robot or provided to it. Figure 3.5 (Koenig et al, 2001) shows the real world in the first picture and the map as seen by the robot in the second picture. The pieces coloured black on the second picture are the aisles and the green coloured pieces are unknown to the robot.

Quasi-static and greedy mapping may be placed in the same category of mapping techniques. The difference between dynamic mapping and these two methods are distinct. Dynamic mapping never actually maps the environment, nor uses path-planning techniques to search through any kind of map for a possible path. On the other hand, quasi-static mapping and greedy mapping do both of these, although it starts with no *a priori* knowledge of the environment.

One of the disadvantages of quasi-static mapping and greedy mapping is that it takes much longer to reach the target position if compared with a system with full *a priori* knowledge. Quasi-static mapping does not store its map after reaching the target position and therefore the whole process is repeated the next time that it is commanded to reach a target position.

The solution to this problem is to allow the system to save its map and then just to update it during successive commands as with greedy mapping. The map will then be complete after some iteration in the same environment. Path planning can then be done from this incomplete map. This process eventually results in static mapping in its purest form.

3.3.4 PATH-PLANNING ALGORITHMS

Before discussing a few path-planning algorithms, it is essential to understand the concept of obstacle expansion. This concept will now be discussed briefly.

The graph method uses hierarchical search of object vertices (corners), to determine a path along these vertices to a goal. Using vertices implies that the robot always moves along the shortest distance around an obstacle to get to the target position. The problem arises that the robot is not point-sized, but has fixed physical dimensions. If it were to move along physical vertices, it would collide with an obstacle at the first vertex. To avoid this problem, obstacle growing is used.

Obstacle growing is the process whereby obstacles are expanded in all directions with the radius of the robot system and the robot's size is decreased to point size. This method is illustrated in Figure 3.6. Growing objects in the physical real world is much more complex than the growing of a discrete memory mapped obstacle. In the real world, it has to be grown first and then the grown obstacle is made discrete. Memory-mapped obstacles are discrete and are grown from there. The growing of a real-world obstacle is slightly more accurate than first making it discrete and then growing it. However, the errors are small compared to the mapping and positioning errors and it is much easier to implement.

When growing a discrete obstacle, it is difficult to grow it equally in all directions, as all distances are discrete. Therefore, obstacles are grown a block less on the diagonals through the obstacle, although the distance is slightly less than the horizontal and vertical growing distances. This is illustrated in Figure 3.7.

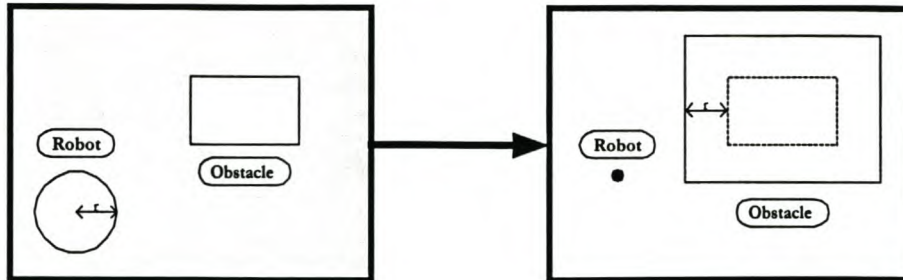


FIGURE 3.6: EXPANSION OF AN OBSTACLE AND SHRINKAGE OF THE ROBOT

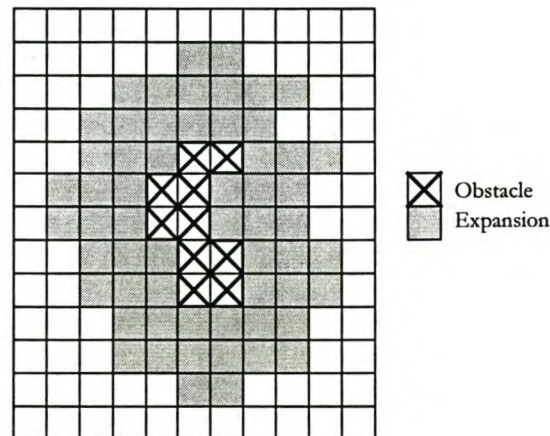


FIGURE 3.7: DISCRETE GROWING OF AN OBSTACLE

Now that the explanation of obstacle expansion has been completed, a few path-planning algorithms will be discussed briefly.

1) VGRAPH Algorithms

The visibility graph is one of the earliest path-planning algorithms. It can be applied to any two dimensional space filled with polygonal obstacles, real world or discrete. The presence of non-polygonal obstacles can be overcome by making the obstacles discrete, therefore using a discrete representation of an environment. The basic principle behind the algorithm is to search a graph of object vertices for the shortest path between two

points. In its purest form, each vertex is assigned a cost and the path of least cost is calculated.

The VGRAPH method uses an *A* searching algorithm*, a graph searching algorithm, which iteratively explores the paths originating from the goal position.

Nodes, or object vertices, are classified as visited or unvisited, so that the algorithm does not search for a path through visited nodes, as its paths have already been covered. In order to perform this search, all the object vertices must be known. The time taken to search the entire graph for all possible paths is a function of N^2 , where N is the number of object vertices.

Off-line path-planning problems have traditionally been approached using either the “mathematical approach” or the “heuristic approach”. The mathematical approach is most concerned with determining an exact solution. The heuristic approach uses known information about the problem’s domain to develop algorithms that are computationally efficient. An exact solution is not guaranteed by a heuristic search.

An *A* search algorithm* is a conglomerate of the mathematical and heuristic approaches. Information from the problem’s domain is strategically incorporated into a formal mathematical approach to reduce the computation time, while yet guaranteeing to find the minimum cost path, if the path exists. Thus, although an *A* search* uses a heuristic approach, it is said to be algorithmic.

The output of an *A* search* is a set of via-points which represents a subset of the nodes in the searched graph. The shortest distance path (SDP) is obtained by connecting the via-points with line segments.

2) Path-planning with obstacle representation in velocity space

Ramírez et al (2001) propose a collision-free path-planner for mobile robot navigation in an unknown environment subject to nonholonomic constraints. This planner is adapted for use with embarked sensors, because it uses only the distance information between robot and the obstacles. The collision-free path planning is based on a new representation of obstacles in the velocity space. The obstacles in the influence zone are mapped as linear constraints into the velocity space of the robot, forming a convex subset that represents the velocities that the robot can use without collisions with the objects.

The planner is composed by two modules, termed “*reaching the goal*” and “*boundary following*”. The major advantages of this method are the very short calculation time and continuous stable behaviour of the velocities.

3) Path Planning Methods in three-dimensional areas

Williams & Jones (2001) presents a method for path-planning in three-dimensional space for an aerial robot. This type of robot is often used to inspect overhead electricity power lines. This algorithm converts a standard three-dimensional array representation of one or more obstacles in the vehicle's environment into an octree and a connectivity graph. Path-planning is then based on a three-dimensional extension of the distance transform.

Two other path-planning algorithms for movement in three-dimensional areas are given by Wang et al (2000). The first one is a potential field approach similar to the one used by Van Rooyen (1997) and the other one a method to convert the path-planning problem into a constrained optimisation problem.

3.4 COMMUNICATION WITH MOBILE ROBOTS

It is possible to control a mobile robot with an onboard control computer or from an external control station. The latter is the preferred method, because of the huge advantages in user-friendliness and ease of use. For this method to be considered, it is, however, necessary to have very reliable communication between the robot and the control station.

In most cases RF communication is used for this. RF senders and receivers are needed for this. A transceiver may be used as well. A transceiver is an RF sender and receiver in one module.

This form of communication is very useful without limiting the control station and the robot to the same room. The electronic or hardware parts as well as the software parts of the system are fairly easily obtained and not too expensive.

3.5 DOCKING TECHNIQUES FOR MOBILE ROBOTS

The docking of a mobile robot involves the entering and stopping of the robot at the destined workstation. The importance of accuracy in this is due to the fact that the gripper must be able to pick up an object from the workstation after docking. If the docking tasks were not completed successfully, the robot would be unable to load and unload items from the different workstations.

The docking problem can be divided into two stages. The first stage, the tracking stage, places the robot in the "line of sight" of the target destination. The second stage, the acquisition stage, involves moving the robot

towards the destination until it has reached the target. The problem can be further subdivided into the following five steps.

- **Tracking:** Place the robot exactly in line with the destination.
- **Orientation:** Turn the robot so that it faces the destination.
- **Acquisition:** Move towards the target in the correct direction.
- **Stopping:** Stop as soon as the robot is in the desired position.
- **Anchoring:** Hold the robot in a fixed position while loading/unloading.

The undocking operation will occur if the same steps are performed in the opposite order. A few docking techniques will now be discussed briefly.

3.5.1 INFRARED TRIANGULATION

A triangulation system consisting of three or more infrared transmitters placed at a fixed distance on the workstation can be used. A rotating sensor on the robot senses the signals and records the angles at which they are sensed. Because the control program knows how far apart the transmitters are and where they are on the workstation, it is possible to calculate the robot's position relative to the workstation.

The advantage of this method is that it uses infrared and will therefore not interfere with the ultrasonic sensors, if there are any on the robot. However, this method on its own is not accurate enough for docking.

3.5.2 WIRE-GUIDED DOCKING

This is a very robust option, which is very easy to implement and uses a wire-guided system. An electric wire or magnetic tape is laid on the floor, leading to every workstation. As the robot nears the specified station, it switches to search mode by turning on magnetic sensors that can sense the wire or magnetic tape. Once it has found the tape, the robot follows the tape, which guides it into the station.

The advantage of this system is its stability. The robot will always follow the wire or tape into the station. If electric wire is used, it might be difficult to change the wire path when the workstation is moved. If magnetic tape is used, the tape path can easily be adjusted.

The inherent problem with this method applies to the acquisition stage of the docking operation. It is very difficult to use a position measuring system in order to keep a robot's orientation correct. A much better option for the acquisition stage is therefore to use a system that measures the angle at which the robot is travelling. Thus, while the wire-guided method may provide a solution to the tracking phase of the problem, it is not considered suitable for the acquisition phase.

3.5.3 *MAGNETIC SENSORS*

A simple option concerning the tracking phase involves the use of a magnet placed on the floor, a fixed distance in front of the workstation. With some research effort, magnetic sensors can be found that, when placed on the robot, will be capable of sensing the magnet from a distance of one metre. Once the robot is directly above the magnet, it will switch over to the sensors used for the acquisition phase.

The great advantage of this system is that the magnet can draw the robot from any direction. Also, once the robot is directly above the magnet, it is certain of its position.

One of the difficulties with this system is the fact that the motors of the robot have magnetic fields around them. This may distort the signal received by the magnetic sensor. This problem could possibly be alleviated by putting the sensor in front of the robot (as far away from the motors as possible) and by shielding it from other magnetic fields.

3.5.4 *INFRARED ACTIVE BEACON*

Instead of the use of triangulation, an infrared, emitter that would emit a single signal, could be placed on the workstation. This would only be a feasible solution for the acquisition phase, as only direction is sensed. Two receivers placed on the front of the vehicle, a fixed distance apart would sense the signal at different strengths. If both signals show the same strength, the robot is moving in the right direction. If one signal strength is smaller than the other signal strength, then the robot is skew and it should turn in the direction of the weaker signal. In this way, the robot will be kept on course.

This is a viable method, which meets the requirements of the target phase of the docking application well. The only problem is that the possibility exists that the system may not be accurate enough.

3.5.5 *MAGNETIC REED SWITCHES*

Reed switches may be a cheap and effective way to accomplish the last step of the docking operation i.e. the stopping action. This would be a more elegant option than simply using contact switches to stop the robot.

3.6 FUZZY LOGIC CONTROL

The most significant difference between fuzzy logic control and conventional control theory is the absence of a complete analytical model of a dynamic system, when using fuzzy control. Fuzzy logic is widely used in industry. One example is an automatic train operation system used in Japan (Berenji 1993). The system uses 12 control rules, is able to exercise constant speed control of the train and stops the train at prescribed locations. The control rules are evaluated every 100 milliseconds and it assures a safe and comfortable ride for the passengers.

3.6.1 BASIC CONCEPTS OF FUZZY SETS AND FUZZY LOGIC

Before the development of fuzzy logic, computers only used crisp sets as input. A crisp set allows either full membership or no membership at all. For example, if the distance to an object d is used as input, and the criterion is 45cm, then the input can only be true or false. This means that the membership value will be one if the input is 45cm, or the membership value will be zero otherwise. In a crisp set, the membership or non-membership of an element x in set A is described by a characteristic function $\mu_A(d)$, where:

$$\mu_A(d) = \begin{cases} 1 & \text{if } d \in A \\ 0 & \text{if } d \notin A \end{cases} \quad \text{EQUATION 2}$$

A fuzzy set is an extension of a crisp set. A fuzzy set allows partial membership that can take values ranging from 0 to 1:

$$\mu_A : D \rightarrow [0,1] \quad \text{EQUATION 3}$$

For the example of the distance to an object, three fuzzy sets can be defined, namely “Very Near”, “Near”, and “Far”, as shown in the following figure. If the distance is 45 cm, then the fuzzy set “Very Near” has the membership value 0.5, that is $\mu_{\text{Very Near}}(45)=0.5$. For the fuzzy sets “Near” and “Far” the membership values are: $\mu_{\text{Near}}(45)=0.5$ and $\mu_{\text{Far}}(45)=0$ respectively.

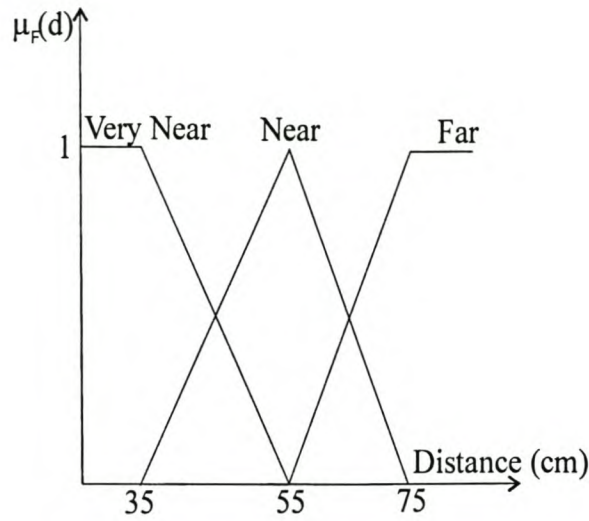


FIGURE 3.8: DISTANCE MEMBERSHIP FUNCTIONS

The variable in the above-mentioned example (distance to an object) assumed the words “Near”, etc. as its values. Variables that can assume words in natural languages (for example, Near, Far) as its values are defined as linguistic variables. These words are usually the labels of fuzzy sets. A linguistic variable can assume either words or numbers as its values; thus the distance to an object can either be “Near” or 50 cm.

Fuzzy logic is based on *If...Then...* rules. Often the situation is encountered when a decision has to be taken on a rule, for example: *If* the object is near, or the object is very near, *then...* It is therefore necessary to define the outcome of situations where the logical operators *and*, *or*, and *not*, are used. The interpretation of *And* in a fuzzy rule is equal to the intersection of the two fuzzy sets. The intersection of fuzzy sets is defined as the minimum of the membership functions, thus:

$$\mu_{Near \cap VeryNear}(d) = \min \{ \mu_{Near}(d), \mu_{VeryNear}(d) \} \quad \text{EQUATION 4}$$

Or is the union of fuzzy sets, and is defined as the maximum of the membership functions:

$$\mu_{Near \cup VeryNear}(d) = \max \{ \mu_{Near}(d), \mu_{VeryNear}(d) \} \quad \text{EQUATION 5}$$

In a statement, for example, *If* d is *not* Very Near, then the membership value is defined as the complement of the membership value for the fuzzy set:

$$\text{d is not very near} = \mu_{\text{VeryNear}}(d) = 1 - \mu_{\text{VeryNear}}(d)$$

EQUATION 6

3.6.2 THE BASIC ARCHITECTURE OF FUZZY LOGIC CONTROLLERS

The basic architecture of a fuzzy logic controller is illustrated in Figure 3.9. The function of the different modules will be described next.

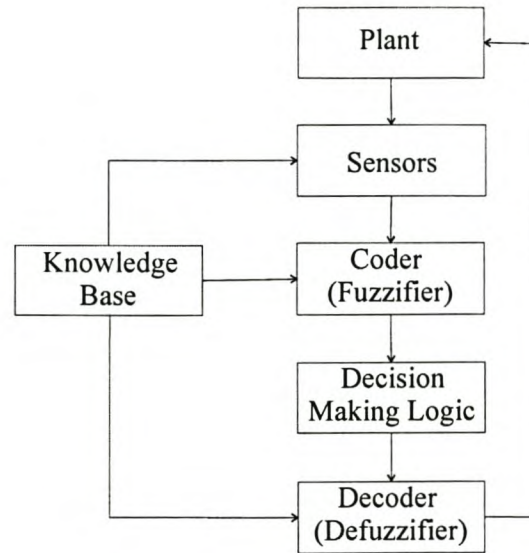


FIGURE 3.9: THE BASIC ARCHITECTURE OF A FUZZY LOGIC CONTROLLER

1) Fuzzifier

The first step in designing a fuzzy logic controller is to decide on the control variables that are going to be used. The variables are obtained from sensors. Sensor readings are normally crisp values. The crisp values are matched against the membership function of the linguistic label. The result is a membership value in the range $[0,1]$.

2) Knowledge Base

There are two main tasks in the design of the control knowledge base. The first is to select a set of linguistic variables describing the values of the main control parameters of the process. It is necessary to define the main input and output parameters linguistically, using the proper term sets. For example: *If the distance to a object is very near, then stop*, where “distance to object” and “stop” are the input and output parameter respectively,

and “near” is the linguistic variable.

The second task is to develop the control knowledge base, which uses the above linguistic description of the main parameters. Four methods for constructing a control knowledge base exist:

- Experts' experience and knowledge
- Modelling the operator's control actions
- Modelling a process
- Self organisation

The knowledge base normally consists of fuzzy *If...then...* rules in the following form:

$$R_n: \text{If } x \text{ (is, is not) } A_n \text{ (and, or) } y \text{ (is, is not) } B_n \text{ (and, or) } \dots \text{ then } z \text{ is } C_n$$

Where n is the rule number, x and y are the input linguistic variables, A_n and B_n are the fuzzy sets, and C_n is the output control parameter. An extension to this rule is a case where the output control parameter is a function of the input parameters, thus $z = f_n(A_n, B_n)$.

The first of the four methods mentioned above is the easiest and most widely used. The second method is effective when expert human operators can express the heuristics of the knowledge that they use in controlling a process in terms of fuzzy rules. Another method is to model the types of control actions taken by the operators, instead of interviewing them. Using the third method, a model of a process is developed and a fuzzy controller is constructed to control the fuzzy model. The approach is thus similar to the traditional approach taken in control theory. The fourth method refers to the research done in developing self-organising controllers. The main principle of this method is the development of rules that can be adjusted over time to improve the controllers' performance.

3) Defuzzifier

The final module of the architecture of a fuzzy logic controller is the decoder or defuzzifier. The function of this module is to produce a non-fuzzy control action from the different fuzzy sets and control rules. Four methods were identified:

- Tsukamoto's method,

- The centre of area method,
- The mean of the maximum method, and
- The method used when the output of the rules is functions of their inputs.

3.6.3 FOUR IMPORTANT ISSUES

Saffiotti (1997) concentrates on four important issues in his paper on “*The uses of fuzzy logic in Autonomous Robot Navigation*”. These issues are:

- how to design robust behavior-producing modules,
- how to coordinate the activity of several such modules,
- how to use data from the sensors, and
- how to integrate high-level reasoning and low-level execution.

4 THE MOBROB PROJECT

The MOBROB research project is one of the research projects currently active at the Department of Industrial Engineering (University of Stellenbosch) and was started in 1993 when Deist (1993) designed fuzzy logic control software for automated guided vehicles. Nel, Mentz and Van Rooyen continued with the project in 1997. Nel designed and built the robot that is now known as MOBROB, while Van Rooyen did the mapping and path-planning and Mentz studied the navigational techniques that can be used in a mobile robot. During 1998, Steenkamp continued the project with his study on communication with a mobile robot using radio frequencies. Woods studied the different docking techniques for mobile robots during 1999. A picture of the robot is shown in Figure 4.1 and a short summary of the different projects mentioned above follows, while the summaries by the different authors are given in APPENDIX A.

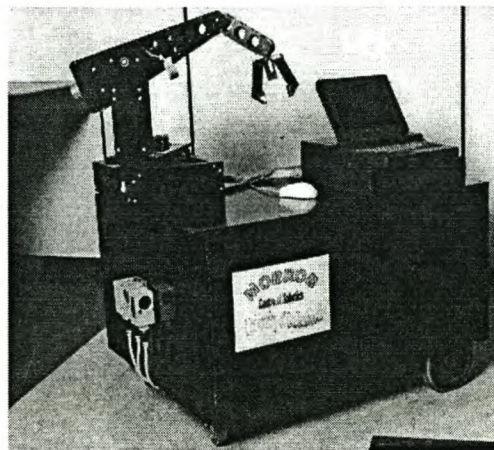


FIGURE 4.1: PICTURE OF MOBROB

4.1 FUZZY LOGIC CONTROLLER FOR AN AUTONOMOUS GUIDED VEHICLE (DEIST, 1993)

Deist (1993) developed a fuzzy logic controller for an AGV as a thesis (Masters degree in Industrial Engineering at the University of Stellenbosch). The knowledge base Deist developed consists of 27 rules. The rules were designed to control the robot in a constant path-width situation and to cope with the following situations:

- Converging to the centre line of a corridor during straight line movement
- Turning left or right
- Turning into junctions (left/right)
- Avoiding obstacles during straight line movement
- Dead-end routes

Deist wrote a simulation program to simulate the different situations as mentioned above. Satisfactory results were obtained for all the situations.

The rules were tested on a model AGV. This AGV is shown in Figure 4.2. The AGV consists of a Perspex base (diameter: 450 mm) with two stepper motors in a differential steering arrangement. Three ultrasonic sensors are mounted on a turret (height: 300 mm). The turret scans through 72° to determine the heading angle of the vehicle, and to measure the East, West, and North distances to objects. The sensors have a measuring range from 50 cm to 6 m with accuracy of 1 cm.

The tests showed that the model is able to converge to the centre line of a corridor during straight-line movement, and is able to cope with dead-end situations by turning around and moving in the opposite direction. Problems occurred during obstacle avoidance and turning.

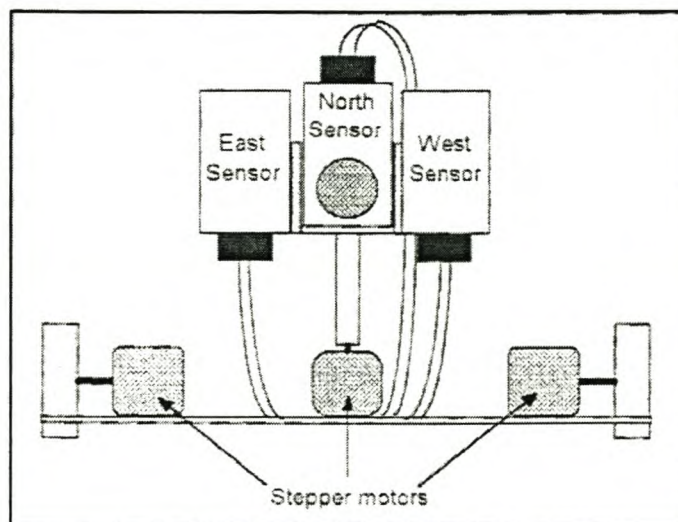


FIGURE 4.2: END VIEW OF A MODEL AGV (NEL, 1997)

Concerning the AGV's difficulty to cope with obstacle avoidance, Deist came to the following conclusion: Obstacles are not supposed to be present in the paths on a factory floor, and if an obstacle occurs, it is fair to assume that it will be only of a temporary nature. He suggested the following control actions when an obstacle is located:

- The AGV has to stop immediately when an obstacle is located.
- The AGV has to wait until the obstacle is removed.
- If the obstacle is not removed after a certain time, the robot has to be re-routed from its present position.

Concerning the robot's difficulty to cope with turning, Deist deduced that a reference track (e.g. two walls) is needed to guide the robot through the turn. To construct paths with constant path widths is physically impossible. Because the rules were designed to guide the AGV to the centre line of a path with a constant width, it meant that Deist would have to redesign the whole knowledge base to enable the robot to make a turn in an environment without constant path widths.

According to Nel (1997), the sensor system yielded problems:

- Electromagnetic interference from the stepper motors resulted in faulty readings from the sensors. Increasing the distances between the sensors and the motors solved this problem.
- Cross interference between the three sensors occurred, which yielded inaccurate measurements.
- The cone of reflection (18°) of the ultrasonic sensors caused the sensors to measure the shortest distance to any object detected within the cone of reflection. This caused orientation errors of as much as 9° (half the size of the sensor cone of reflection), which in turn led to substantial positional errors.

4.2 THE DESIGN AND CONSTRUCTION OF A MOBILE ROBOT FOR MATERIAL HANDLING (NEL, 1997)

According to Nel (1997) the components of a mobile robot can be categorised into two groups, namely hardware and software components. In the following paragraphs the hardware and software used by Nel (1997) are discussed in more detail.

4.2.1 MOBROB HARDWARE

The hardware Nel (1997) used in MOBROB consisted of:

- The vehicle,
- Batteries,
- Motors and gearboxes,
- Servo power amplifiers,
- Control computer,
- Control card,
- I/O card, and
- Ultrasonic sensors.

4.2.2 MOBROB SOFTWARE

The software used to control MOBROB can be categorised into three groups, namely, the software supplied with the control card, programmes written in TURBO PASCAL and Quinn-Curtis real time tools.

4.3 THE MAPPING AND PATH-PLANNING OF AN INDUSTRIAL MOBILE ROBOT (VAN ROOYEN, 1997)

Van Rooyen studied the different algorithms available for path planning of a Mobile Robot. Three methods were discussed and evaluated. A workable, functional algorithm and software to illustrate the working of the algorithm were developed. This algorithm is a fresh way of approaching the path-planning and mapping of a mobile robot and will now be discussed thoroughly.

4.3.1 OPTICAL ROBOTIC POTENTIAL FIELD MAPPING

- The optical robotic potential field mapping system has an imaging device and a processor.

- Two image-writing modes are used by the imaging device: electron deposition and electron depletion.
- Patterns written in electron depletion mode are sharp and appear white. The generated image represents a robot's workspace.
- The imaging device under processor control then writes a goal location in the workspace using electron deposition mode where the black image of the goal expands in the workspace.
- The processor stores the generated images and uses them to generate a feedback pattern.
- The feedback pattern is written in the workspace by the imaging device in the electron deposition mode to enhance the expansion of the original goal pattern.
- After the feedback pattern has been written, an obstacle pattern is written by the imaging device in the electron depletion mode to represent the obstacles in the robot's workspace.
- The processor compares each stored image to the previously-stored image to determine any differences between them.
- After the output image has stopped changing, the computer stores a sequence of output images. In the first image, only the goal location is dark. In the last image, only obstacles are bright.
- An average is determined from all stored images. The average that results from the algorithm is the desired potential field map.
- A robot placed at any location in the workspace represented by the potential field map will be able to follow the gradient of the potential to the goal location, without being trapped behind obstacles.

4.3.2 PRESENTATION OF THE FACTORY

The factory is divided into a fixed block-sized grid. The size of the block determines the resolution, and the size of the grid, the area to be converted and the environment to be mapped. As the robot is unlikely to encounter too many hills, it is assumed that the co-ordinate system is two-dimensional.

A potential field map of a bounded, two-dimensional region containing a goal location and arbitrary number of obstacles is presented. A mobile robot can guide itself from any location to a goal location, while avoiding all obstacles present by using the potential field map description of the region. The workspace is modelled as a

collection of electric charges and their associated potential fields. The robot is modelled as a point charge with the same charge as the block on the grid it is standing on. Its destination is a negatively-charged region, while regions of strong positive charge represent obstacles.

Path-planning is a matter of working out the route over which the potential value between the robot and neighbouring blocks drops fastest.

4.3.3 FIELD ASSOCIATED WITH CHARGES

Consider the potential associated with a point charge at the origin of a spherical co-ordinate system. The integral form of Gauss' law obtains the electric field around the point: Gauss's law describes how the electric field intensity is related to its source. The net charge within an arbitrary volume V that is enclosed by a surface S is related to the net electric flux through that surface by:

$$\oint \epsilon_0 \cdot E da = \int \rho dv \quad \text{EQUATION 7}$$

Where ϵ_0 = the permittivity of free space, 8.854×10^{-12} farad/metre

S = surface

$\epsilon_0 \cdot E$ = electric displacement flux density

It follows from the definition of the potential that the potential of a point charge is:

$$\Phi = \frac{q}{4\pi\epsilon_0 r} \quad \text{EQUATION 8}$$

Where q = the potential of a point charge

R = the scalar distance between the point of observation and the charge

By plotting the values of these lines of electric fields on a z-axis, a three dimensional view of a mountainlike factory can be created. This is illustrated in Figure 4.3.

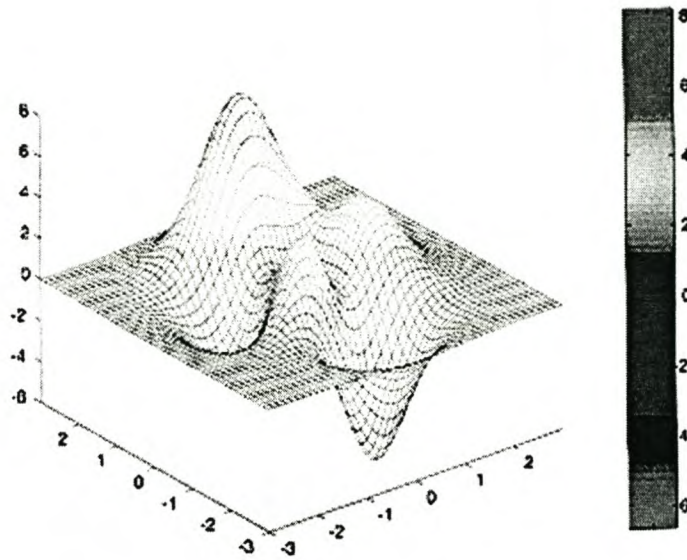


FIGURE 4.3: A 3D PLOT OF ELECTRIC FIELD LINES

It is important to take the superimposing property of the field lines into account. Two obstacles very close to each other will have the effect of a single obstacle, because of the added values of the field lines that cross each other. This will automatically prevent the robot from moving into too narrow openings.

4.4 A STUDY OF THE DIFFERENT NAVIGATIONAL TECHNIQUES FOR A MOBILE ROBOT (MENTZ, 1997)

Mentz studied various different methods for the navigation of a Mobile Robot. After completing this, two practical, workable, easy to implement methods were chosen for further evaluation. This evaluation was completed and one of the methods was chosen for implementation. This method will now be discussed in detail.

4.4.1 DETAILED DESCRIPTION OF THE NAVIGATIONAL METHOD DEVELOPED BY MENTZ

- In every top corner of the room, a separately-driven 555-timer is placed, driving an infrared LED.
- All four transmit simultaneously and continuously.

- The differences between them, that will ultimately determine the robot's unique location, are the different frequencies with which they are transmitting (Figure 4.4)
- The infrared detector on the robot is mounted on the stepper motor and turns with it.
- The position of the stepper motor is recorded when the detector receives a signal.
- The infrared receiver samples the received signal at a sufficiently high frequency, in this case, 8kHz.
- A counter counts the number of 8kHz pulses fitted into the received signal and through calculation can determine the identity of the current corner.
- The number of steps from the stepper motor is counted between observations, and with this information, the angle between different corner transmitters is calculated.
- It is now possible to identify every corner uniquely and with this information, the robot's position can be determined. The minimum number of corners the robot needs to see before its position can be calculated, is three.

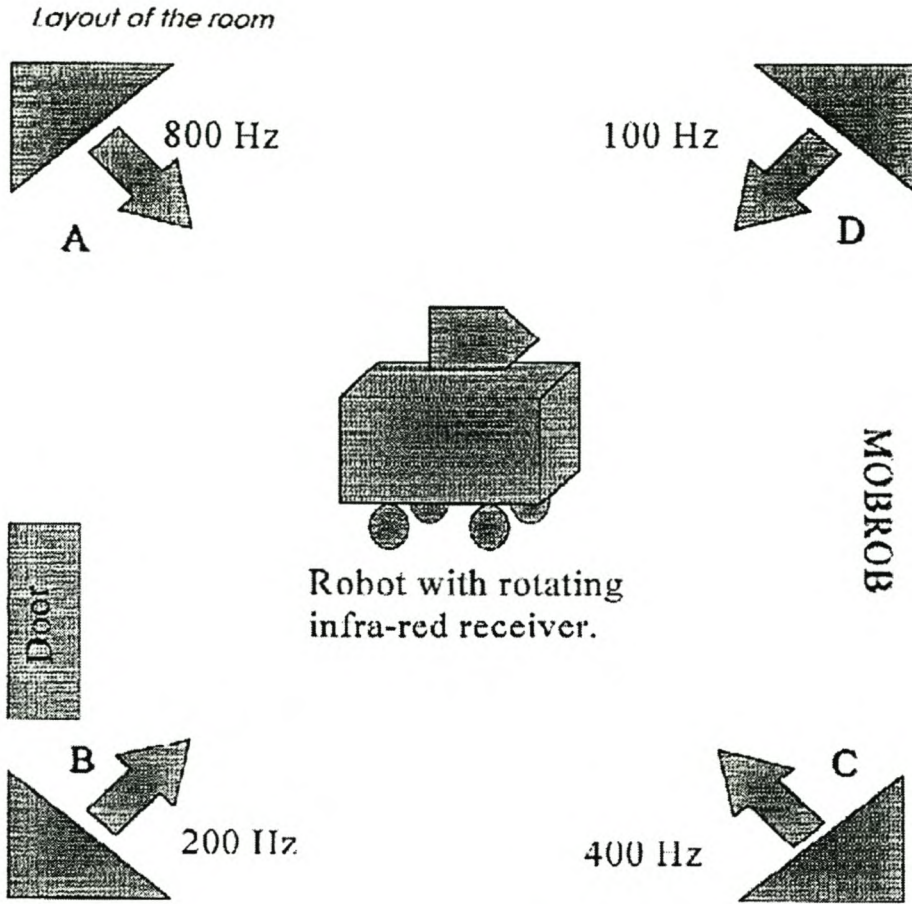


FIGURE 4.4: BASIC CONFIGURATION OF THE SYSTEM

The mathematical calculations for this method are given in APPENDIX B. The final formulas follow:

$$x = \frac{a(1 + \cot \alpha \tan \gamma) \tan \gamma}{1 + \tan^2 \gamma} \quad \text{EQUATION 9}$$

$$y = \frac{a(\tan \gamma - \cot \alpha) \tan \gamma}{1 + \tan^2 \gamma}, \text{ where} \quad \text{EQUATION 10}$$

$$\tan \gamma = \frac{b(\cos(\alpha + \beta) \sin \alpha}{b(\sin(\alpha + \beta)) \sin \alpha - a \sin \beta} \quad \text{EQUATION 11}$$

The values of a , b , x , y , α , β and γ are indicated on Figure 4.5.

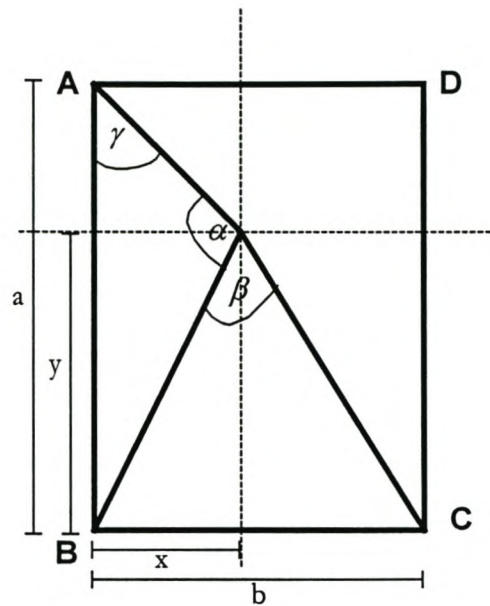


FIGURE 4.5: PARAMETERS USED IN CALCULATIONS

What is needed for the above calculations?

- Sides a and b should be known, which is true for all calculations, since the workspace of the robot is a closed room with known dimensions.
- Angles α and β are known from measurements done by the stepper motor.

Therefore, the input for the system will be: a, b, α, β .

The desired output is: x, y and direction that the robot is facing in terms of positive/negative x/y direction.

There are a few different existing scenarios in the calculation of the robot's position. For explanatory purposes, one of these scenarios is included in this document. For more information Mentz's (1997) document may be consulted (Mentz, 1997, Paragraph 7.3 4 Different scenarios that can exist). This scenario (called scenario 0) is used when the infrared detector sees corner A first, while moving in an anti-clockwise direction. The first angle recorded between corner A and B is called α , and the second angle between corner B and C is called β . This is shown in Figure 4.6: Scenario 0

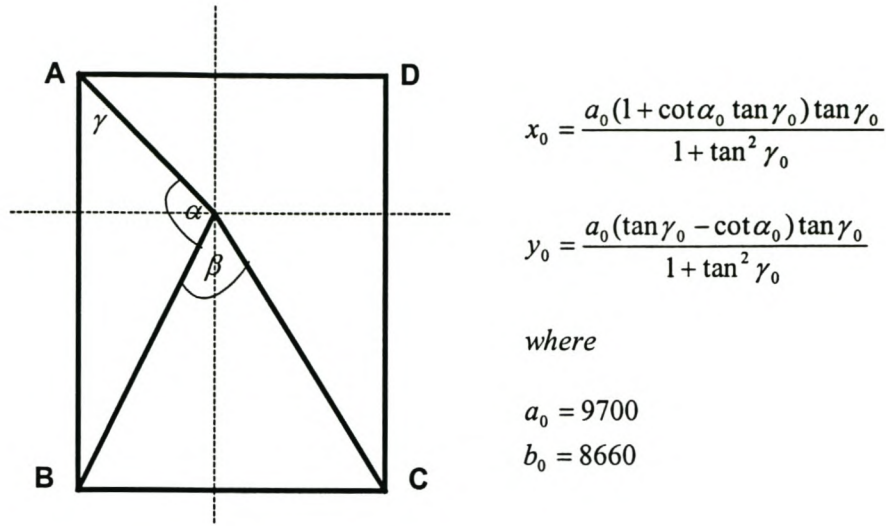


FIGURE 4.6: SCENARIO 0

All the other scenarios can be described in terms of Scenario 0.

4.4.2 THE DIRECTION THE ROBOT IS FACING

It can be described as the direction in which the front of the robot is facing. The long axis of the robot is inclined at a certain angle. The purpose is to determine this angle.

Two sub-scenarios can exist inside each scenario. If a new co-ordinate system can be placed with its origin on the x, y co-ordinates computed from the position of the robot, the direction in which it faces can be calculated as an angle deviation from the positive x-axis. This is explained in the following two figures.

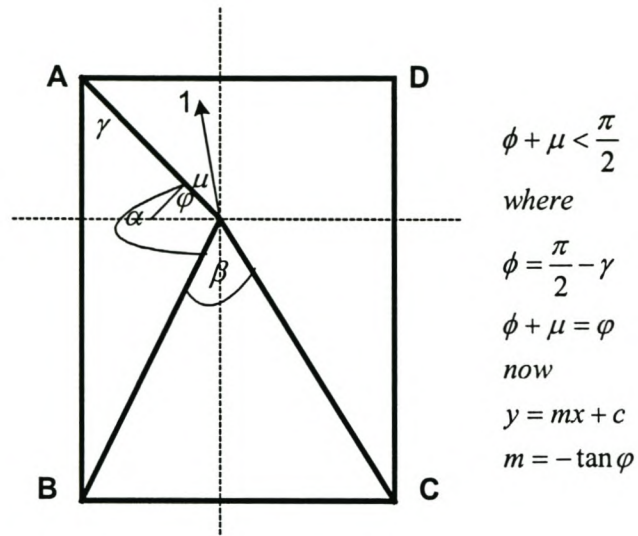


FIGURE 4.7: SUB-SCENARIO 0.1 FOR DIRECTION OF FACING

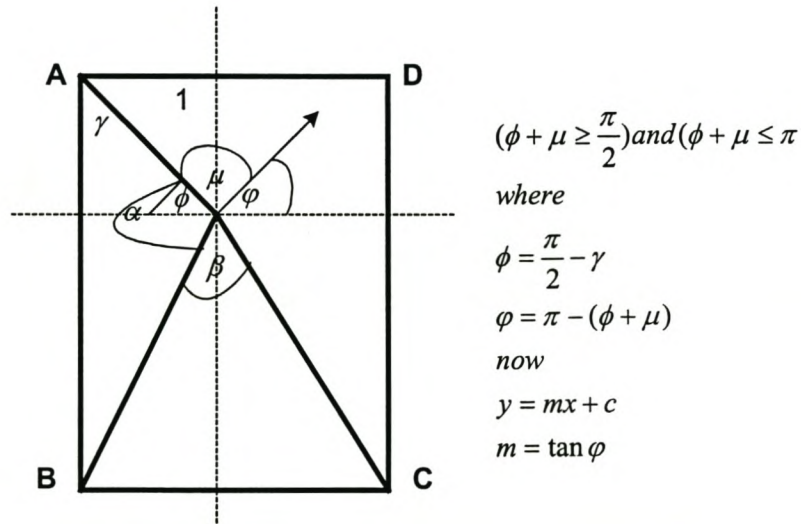


FIGURE 4.8: SUB-SCENARIO 0.2 OR DIRECTION OF FACING

To determine the direction in which the robot is facing, a few assumptions can be made:

- The robot is facing in the same direction as the stepper motor's starting position.
- In this case it will be assumed that the infrared detector cannot miss a transmitted signal when moving past it. It may occur in the practical application, though. It is possible to compensate for

this error without any trouble and this should not be a determining factor in the calculations.

- The angle between two received signals cannot be greater than 180° , but that is also based on the previous assumption.
- The angle from the starting position of the stepper motor, i.e. the direction the robot is facing, to the recognition of the first signal, is called μ .

As each scenario can be divided into two smaller scenarios for the purpose of determining the direction the robot is facing, the assumptions made for these calculations must always be kept in mind. In all cases, μ is the angle between the direction in which the robot is facing and the first signal it receives.

In both Figure 4.7 and Figure 4.8, a little mathematical manipulation can yield the equation for the line on which the robot's long axis is lying. The input to these manipulations are x and y , obtained from previous calculations.

4.4.3 PARTS OF THE SYSTEM

The system consists of two parts, i.e. the transmitter and the receiver.

Transmitter Part

In every top corner of the room, a unit similar to the one shown in Figure 4.9 is placed. The 555-timers generate the signals to pre-set frequencies, in this case 100, 200, 400, and 800 Hz. The LED's transmit these frequencies. 9V batteries can power the four transmitters. This makes the system fully portable and flexible. However, the option to power the transmitters with ac power for permanent systems is also required. This option is not possible with Mentz's system.

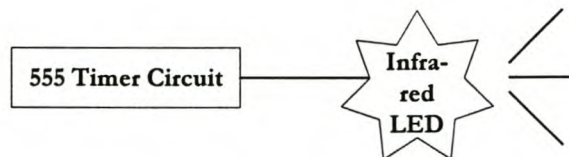


FIGURE 4.9: THE TRANSMITTING PART OF THE SYSTEM (MENTZ, 1997)

Receiver Part

The infrared detector receives the frequency-unique signal from one of the corners of the room. The device

used for this function is a standard device. The signal is then sent to the sampler that samples at a certain frequency high enough according to the Sampling Theorem stated below:

"A real-valued band-limited signal having no spectral components above a frequency of B Hz is determined uniquely by its values at uniform intervals spaced no greater than $1/(2B)$ seconds apart."
(Stremmer, 1990)

According to this theorem, a frequency that is suitable for all four transmitted frequencies should be found, that is, a frequency high enough so that the distinction between the different transmitted frequencies can be seen clearly. The chosen frequency for this system is 8kHz. The counter counts the number of sampler oscillations fitted into one sampled oscillation. The counts for the different transmitters follow:

$$100\text{Hz Transmitter} \quad 8000/100 = 80$$

$$200\text{Hz Transmitter} \quad 8000/200 = 40$$

$$400\text{Hz Transmitter} \quad 8000/400 = 20$$

$$800\text{Hz Transmitter} \quad 8000/800 = 10$$

The stepper motor used has 200 steps per rotation. Therefore,

$$1 \text{ STEP} = 15.708\text{E-}03 \text{ rad} = 0.9^\circ.$$

A visual representation of the receiver part of the system follows:

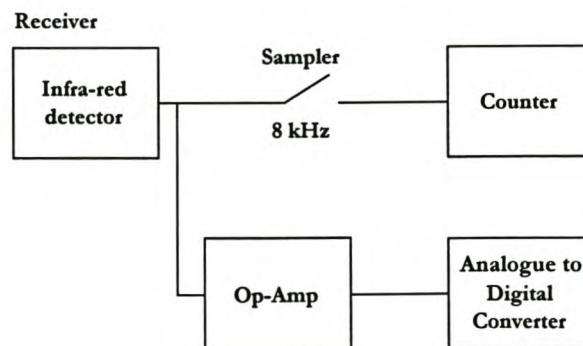


FIGURE 4.10: THE RECEIVING PART OF THE SYSTEM (MENTZ, 1997)

Mentz concluded that:

- An exact (x; y) co-ordinate for the position of the robot can be calculated, given a, b, α , β .
- This system is flexible, as it is independent of any changes made in the environment. If obstacles are added, it would only have an effect on the path-planning, not on the positioning system. The system can also be moved to any other room with 90° corners. The length: width ratio need not be the same.
- The system cannot function if the room does not have four 90° corners.

4.5 COMMUNICATION WITH A MOBILE ROBOT USING RADIO FREQUENCIES (STEENKAMP, 1998)

This subset of MOBROB can again be divided into hardware and software components.

4.5.1 COMMUNICATION HARDWARE

Two Radiometrix radio-frequency transceivers are used. These transceivers operate with a centre frequency of 433 MHz. This frequency is in the ISM band.

In Steenkamp's solution, the transceivers were controlled with the parallel port of the computer and the output of the computer was buffered with an IC. The data (received from)/(sent to) the transceiver went through the serial communication port of the computer. These lines were buffered with a MAX232 chip. This chip transfers the -12V high of the RS232 communication port to 5V and the +12V low of the RS232 communication port is transferred to 0V. A 0V low and a 5V high are required for the transceivers.

Transceivers

The pin settings for the Radiometrix transceivers are given in Table 4.1 including a brief description of each pin's function.

TABLE 4.1: PIN SETTINGS FOR THE RADIOMETRIX TRANSCEIVERS

PIN #	FUNCTION	DESCRIPTION
1 & 3	RF Ground	Antenna's ground, internally connected to 9, 10, and 18.
2	Antenna	16.5 cm WIP antenna.
9, 10 & 18	Vss	0V for modulation.
11	CD	Carrier detect.
12	RXD	Digital receiving signal.
13	RX Audio	Analogue receiving signal.
14	TXD	Digital sending signal.
15	TX select	0V for sending mode, else 5V.
16	RX select	0V for receiving mode, else 5V.
17	Vcc	5V positive supply.

Both circuits must be able to send and receive data. Steenkamp controlled this with inputs from the parallel port of the computer to pins 15 and 16 of the Radiometrix transceivers. In Table 4.2, 1 is equal to an input of 5V and 0 is equal to an input of 0V.

TABLE 4.2: CONTROLLING THE SENDING/RECEIVING MODES OF THE TRANSCEIVER

Pin 15 TX	Pin 16 RX	Function
1	1	Power off
1	0	Receiver mode
0	1	Sender mode

0	0	Self test control loop
---	---	------------------------

MAXIM232

The serial communication port of the computer is used for the sending and receiving of data to and from the transceiver. The transceiver operates with voltages between 0V and 5V. 0V is a binary low and 5V is a binary high. The RS232 communication port, uses -12V for a binary high and 12V for a binary low. To integrate these different voltages, an RS232-interface-IC, the MAXIM232 is used. In Figure 4.11 a typical circuit diagram for the MAXIM232 IC is shown.

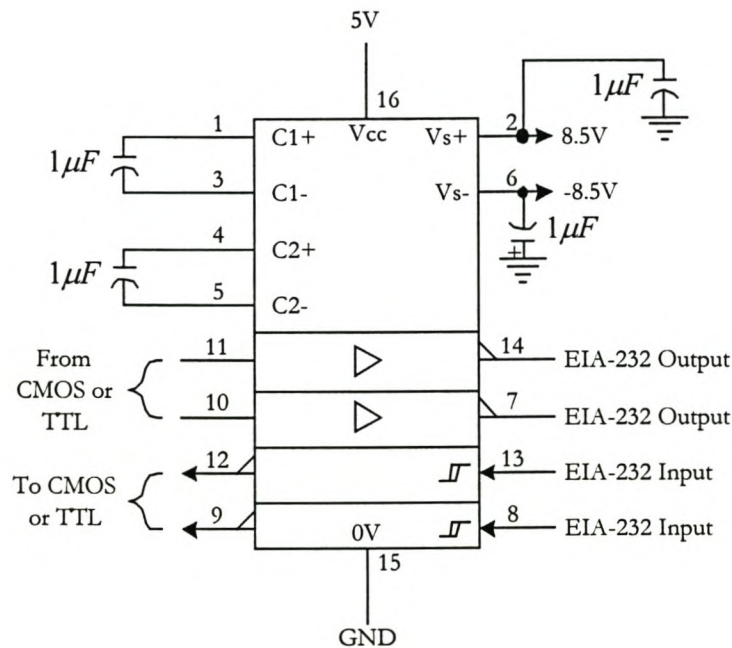


FIGURE 4.11: TYPICAL MAXIM232 CIRCUIT DIAGRAM

5V Regulator

A 5V regulator for the onboard transceiver circuit is used to bring the available onboard 12V (car battery) down to the needed 5V. This is shown in Figure 4.12: 5V Regulator

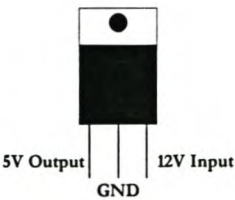


FIGURE 4.12: 5V REGULATOR

4.5.2 COMMUNICATION SOFTWARE

Steenkamp wrote some modules in TURBO PASCAL 7. All these modules were grouped in a unit, namely RF.tpu. This unit was added to the MOBROB main program.

In Table 4.3 a description of all the procedures and functions of the RF.tpu unit is given.

TABLE 4.3: PROCEDURES AND FUNCTIONS IN THE RF.TPU UNIT

Procedure name	Description
Init	Initialising the transceiver circuit. Baud rate is set and the circuit is set up in sending or receiving mode.
OntvangArray (Receive Array)	Receive an array. The transceiver circuit is in receiver mode and is waiting to receive an array. Start and stop bits will indicate the beginning and end of the array.
StuurArray (Send Array)	Send an array. The transceiver circuit is in sending mode and is sending an array with the necessary start and stop bits included.
OntvangLer (Receive File)	Receive a file. The transceiver circuit is in receiving mode and waiting to receive a file. The file is then saved on the local hard drive.
StuurLer (Send File)	Send a file. The transceiver circuit is in sending mode, the computer reads a file from the local hard drive, adds the necessary start and stop bits, and sends the file continuously.
Enkodeer (Encode)	Encode data in array format before sending it with StuurArray.

Dekodeer (Decode)	Decode data received in array format.
ReceivePath	Place the program in a waiting cycle to receive route. Once the route is received, save it in a file.
ReceiveMap	Place the program in a waiting cycle to receive a map. Once the map is received, save it in a file.

Protocol

The RS232 protocol is used for the sending and receiving of data (Table 4.4). The serial communication port is addressed directly.

TABLE 4.4: SUMMARY OF THE PROTOCOL USED IN DATA SENDING AND RECEIVING

Parameter	Address	Value
Baud rate	3F8 & 3F9	9600
Parity	3FB	No parity
Byte length	3FB	8 bits
# Stop bits	3FB	2 stop bits

The first few bytes sent have some errors in them. Therefore a few start bytes are used. This, along with the end bytes must be removed from the received data. The end bytes are used to indicate the end of the data.

5 LITERATURE CONCLUSIONS

On completion of the literature review, certain conclusions were made. These conclusions are discussed briefly in this chapter.

5.1 GENERAL CONCLUSIONS

The first chapter in this review on Computer Integrated Manufacturing may seem unconnected but in this chapter the author tries to illustrate that the complexity of an integration problem increases very rapidly.

5.2 COMPUTER INTEGRATED MANUFACTURING

It was found that the problem of integration is not trivial and that the complexity of this group of problems can increase rapidly. It is therefore important to design components for integration and not to leave this task for the completion of a project.

5.3 MOBILE ROBOTICS

The literature review on mobile robotics was done with the MOBROB project in the AS-IS state in mind and the author mainly concentrated on the mobile robotic issues as related to the MOBROB project. The reason for this is that the author had the task of integrating the existing subsets and not developing new ones. The author therefore needed all the information on the existing and related technologies.

Articles by Borenstein were found extremely relevant to the MOBROB project.

Some of the methods used can be seen as “old technology” due to the fact that the MOBROB project has been running for a number of years.

5.4 THE AS-IS MOBROB PROJECT

Only one of the subsets of the MOBROB project was working in its entirety when the author started with the integration project. The program codes available to the author were faulty and had to be debugged and even

rewritten in some cases. Some of the hardware had to be redesigned and rebuilt.

The author found these tasks very time consuming and unnecessary.

5.5 GENERAL REMARKS

It is very important to integrate a project throughout its life cycle and not only at the end of the design phase. Errors or inability to integrate is time consuming and expensive.

Proper communication and documentation can ease the integration process and save time and money. This project is an example where the lack of proper documentation was found to be the major problem. This point cannot be emphasised enough if the project is divided by time and people as well.

SECTION THREE: THE INTEGRATED CONTROL SYSTEM AND ITS COMPONENTS

This section describes the integrated control system in general as well as its software and hardware components.

6 THE INTEGRATED CONTROL SYSTEM

In this chapter an overview of the integrated control system developed to control MOBROB is given. The separate parts of the control system will be discussed in detail in the two following chapters. All the figures referred to in the text are included at the end of the chapter and all program codes are included in Appendices D, E and F.

6.1 GENERAL

At this stage, the author would like to refresh the reader's memory. The objective of this project was to develop an integrated control system for the subsets of a mobile robot and to implement this control system on MOBROB.

The existing subsets in this project were:

- The Vehicle,
- A Fuzzy Logic Controller,
- Global positioning system,
- Path-Planning system, and
- RF Communication hardware and software.

The control system must be designed so that: The existing subsets work in perfect harmony and new subsets can easily be attached to the control system in the future.

The subsets developed in the past were a given and the previous developed software were used as far as possible. However, in all the cases, it was not possible to integrate without changing the software to some extent.

The flow diagram given in Figure 6.1 illustrates the general working of the control system. The system starts when the base station control program is running. This program was written in DELPHI.

The four diamond-shaped blocks at the top of Figure 6.1 represent the four starting options the user has.

These options are to:

- ➡ Create new maps, edit maps or do path-planning,
- ➡ Do quick path-planning,
- ➡ Send a chosen path to MOBROB, and
- ➡ Send a quick planned path to MOBROB.

If one of the first two options is chosen, MATLAB is opened and the path-planning or quick path-planning programs are opened. Once the user has finished working with the path-planning or quick path-planning programs, the “MATLAB” and “figure Nr 1” windows must be closed.

The user is then back at the four options. The other two options, the sending and quick sending options send a path to the robot using the serial communication port (COM1) of the computer. Connected to the COM1 is an RF circuit with a Radiometrix chip. The path consists of two files, the *.map and *.pth files. The *.map and *.pth files are created by the path-planning programs and are in the correct format to be understood by MOBROB.

MOBROB receives the path via the serial communication port (COM1). MOBROB then simulates the path from these two files. The user is reminded to switch the driving motors and sensors on and MOBROB continues towards its destination. Every time MOBROB has to make a turn, a node exists and MOBROB's position is updated at this stage. At present, the user has to enter the three corners required by the triangulation system on the onboard computer. Once the position has been updated, the robot will continue to the next node or turning point until its destination is reached. At this stage, the position is again updated and

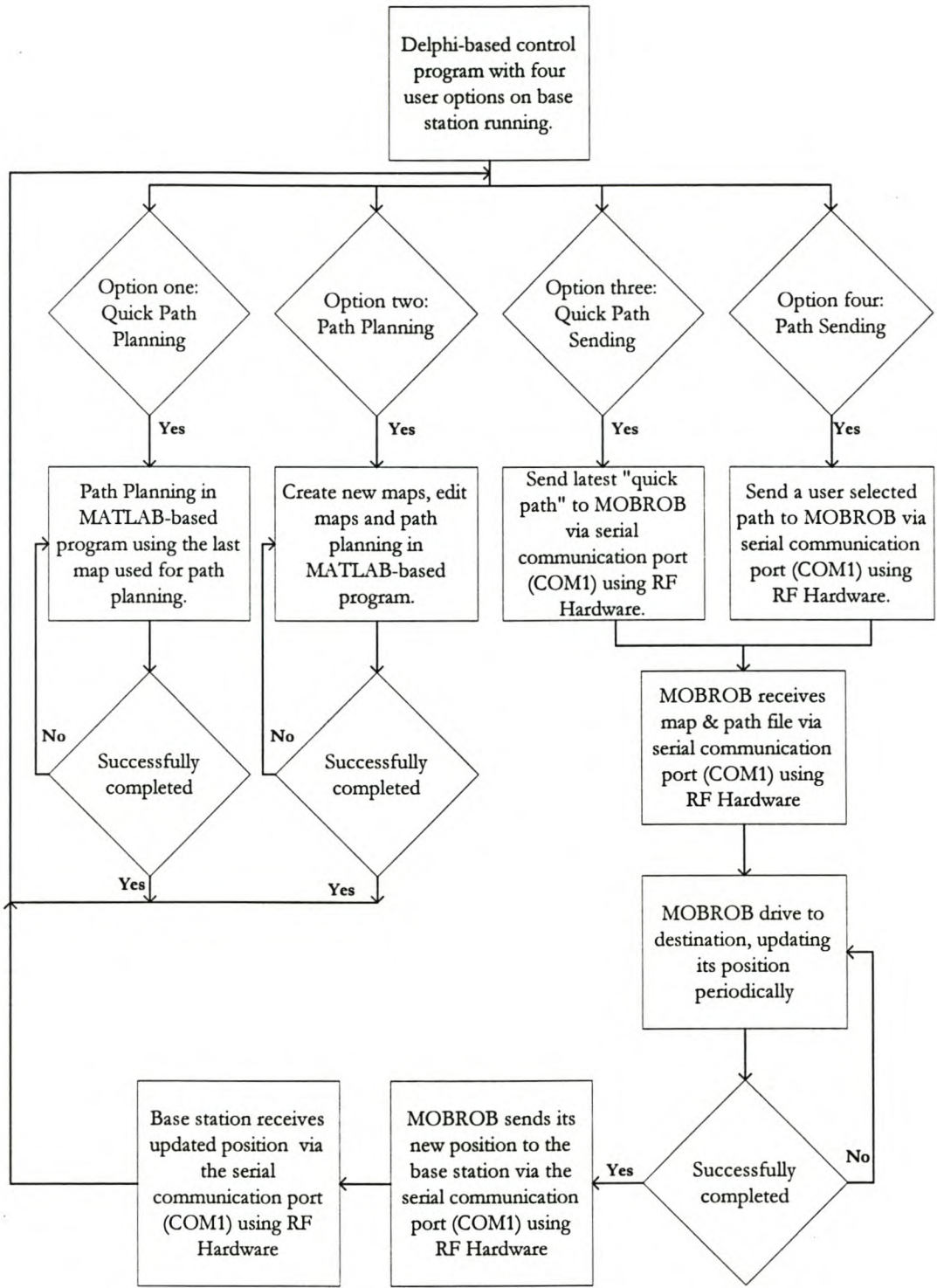


FIGURE 6.1: FLOW DIAGRAM OF CONTROL SYSTEM

saved in current.pos. Current.pos is sent back to the base station, again using the RF communication system.

The new `current.pos` file received by the base station replaces the previous position and the user can plan the next path for MOBROB.

This cycle can then be repeated.

6.2 OVERVIEW OF THE DELPHI-BASED PROGRAMS

The DELPHI-based program (Figure 6.2) is the main controlling program on the base station. The user still has the same four options to start with.

If the user chooses one of the two path-planning options, control is handed over to the MATLAB-based path-planning programs. When the user has finished the path planning, control is handed back to DELPHI and the user again has the four options.

If one of the sending options are chosen, the two path files (*.map and *.pth) are sent to MOBROB. The Radiometrix chip needs to be initialised by sending the binary string '10101010...' or the character string 'UUU...' for a few microseconds. The files are therefore encoded to include this initialising string, a starting string and a stop string. MOBROB will then know that the data will follow directly after the starting string and that the data stopped just before the stop string.

The program will wait in receive mode for MOBROB to return its new position to the base station. Once this data is received, it is decoded (removing the initialising string, start string and stop string from the received data) and saved in `current.pos`, replacing the previous file.

Once again, the user now has the four original options.

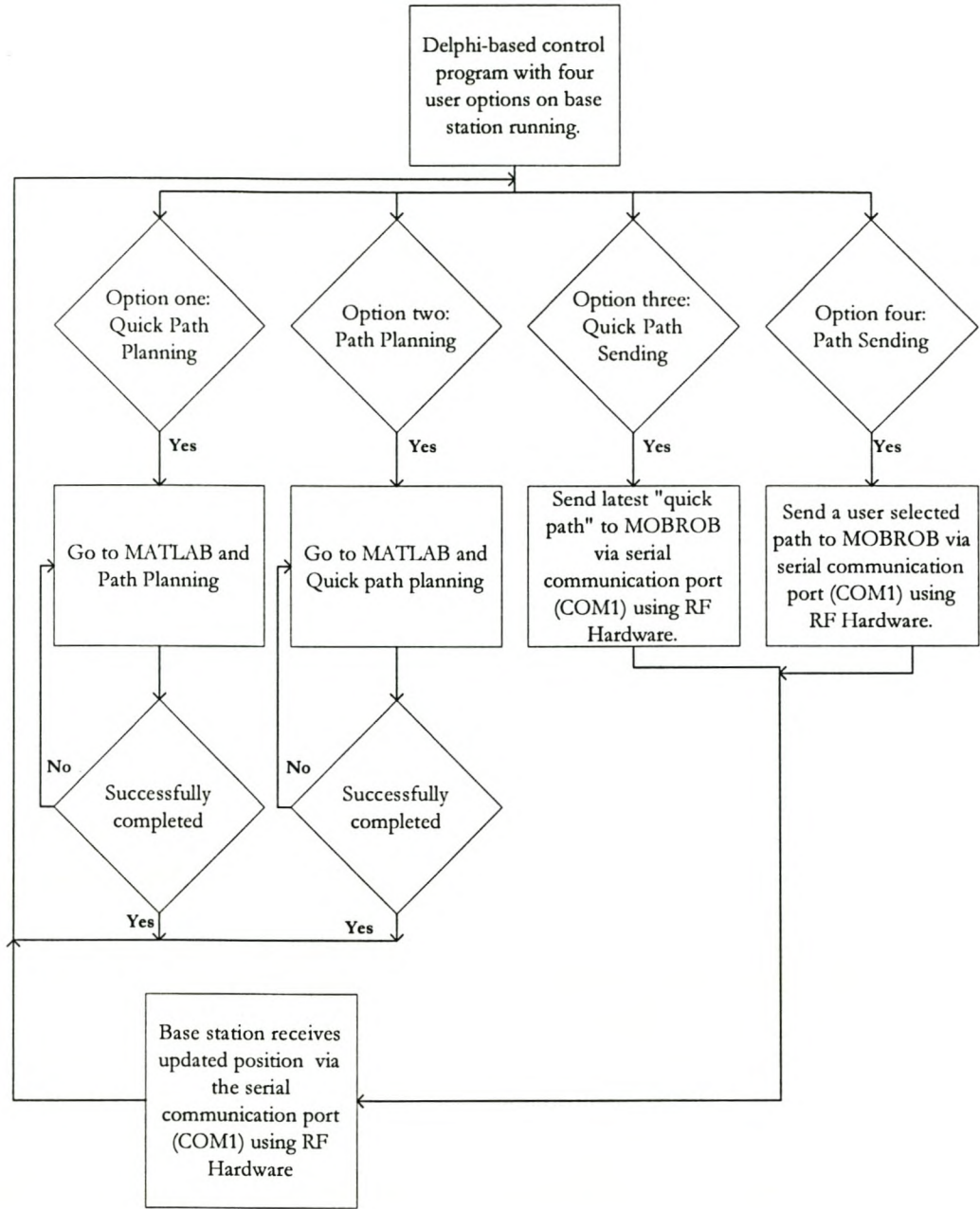


FIGURE 6.2: FLOW DIAGRAM OF DELPHI-BASED BASE STATION CONTROL PROGRAM

6.3 OVERVIEW OF THE MATLAB-BASED PROGRAMS

There are two MATLAB-based programs. The first one is used for quick path-planning and the second one for creating and editing maps. It is possible to do path-planning with the second program as well.

6.3.1 OVERVIEW OF THE QUICK PATH-PLANNING PROGRAM

The quick path-planning program (Figure 6.3), that will be used more often, loads the previously-used MATLAB map file, saved in `wmap.mop`, and MATLAB potential field file, saved in `wmap.pot`, along with the current position of MOBROB, `current.pos`. The `current.pos` file consists of the x and y co-ordinates of the robot, along with the direction the robot is facing.

The user can now select a destination on the map (visible on the computer screen) using his/her mouse.

Once a valid destination has been entered, the program will plan a path for MOBROB. The path, consisting of “corner co-ordinates” or co-ordinates where the robot must turn, is altered and saved in the two path files MOBROB requires, namely `wmap.map` and `wmap.pth`. The different file formats of the files will be discussed in Chapter seven.

6.3.2 OVERVIEW OF THE PATH-PLANNING PROGRAM

The second path-planning program (Figure 6.4) will be used mainly for creating a new MATLAB map file and editing existing MATLAB map files (*.map).

When creating a new map, the user enters the length (maximum x) and the width (maximum y) in metres. A map is created with a wall on the outside. The user can add obstacles with the mouse. Once all the obstacles have added, the user clicks on the red triangle in the bottom right corner of the map (on the screen). The potential field map is created and the user is asked for a filename. The files (*.mop and *.pot) are then saved.

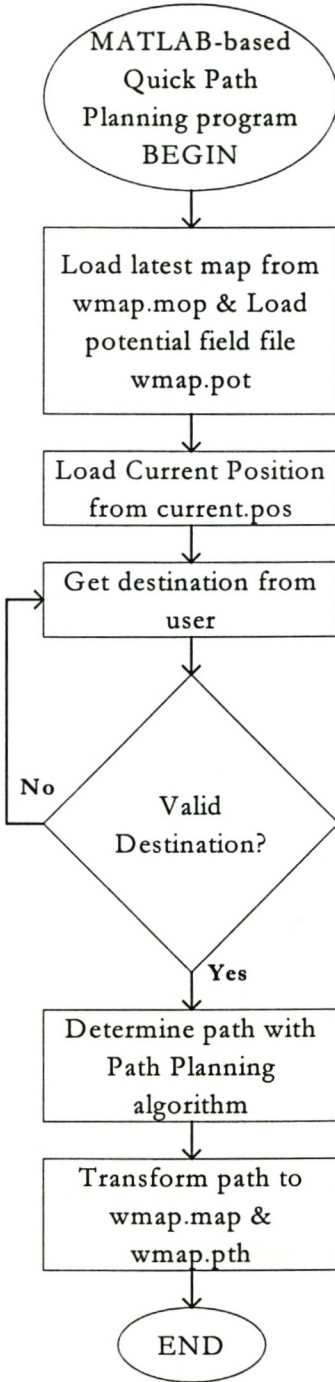


FIGURE 6.3: FLOW DIAGRAM FOR THE MATLAB-BASED QUICK PATH-PLANNING PROGRAM

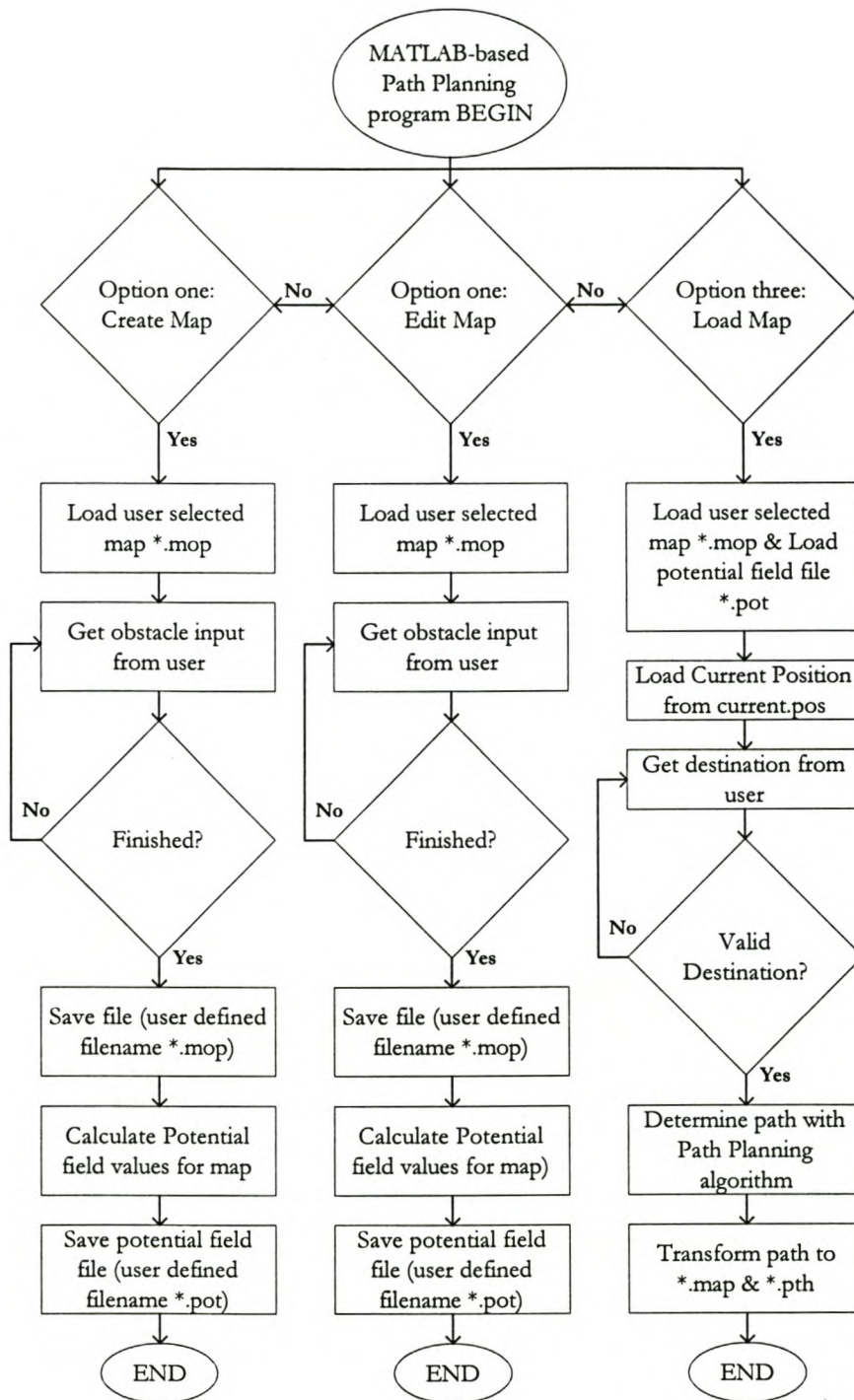


FIGURE 6.4: FLOW DIAGRAM FOR MATLAB-BASED PATH-PLANNING PROGRAM

If the user wants to remove an obstacle, the co-ordinates can be clicked with the mouse for a second time, and the obstacle is removed.

When a map is edited, the user is asked for the filename of the map he/she wants to edit. The map is drawn on the screen and the user can add and remove obstacles using the same method as before. Once the user has finished the editing, the potential field map is created and the user is asked for a filename. The maps are saved.

The first time a path is planned for a new map, it must be done here. The first step in this process is to load the MATLAB map file. The user selects the map that must be loaded.

The user can now select a destination on the map using his/her mouse.

Once a valid destination has been entered, the program will plan a path for MOBROB. The path is formatted and saved in the two path files MOBROB requires, namely *.map and *.pth.

6.4 OVERVIEW OF THE PASCAL-BASED PROGRAMS

MOBROB's onboard computer uses the Pascal-based programs to control the MOBROB hardware. Since the ideal situation is for MOBROB to be completely controlled by the base station, the user inputs in these programs were minimised. The flow diagram for the main program is given in Figure 6.5. An overview of the working of this program is also described in the following paragraphs.

The first step completed by these programs is to initialise the RF communication hardware, put it in receiver mode and wait for the path files to arrive.

When the path files arrive, they are saved on the computers C drive. The current.pos file is opened to obtain the current position of MOBROB.

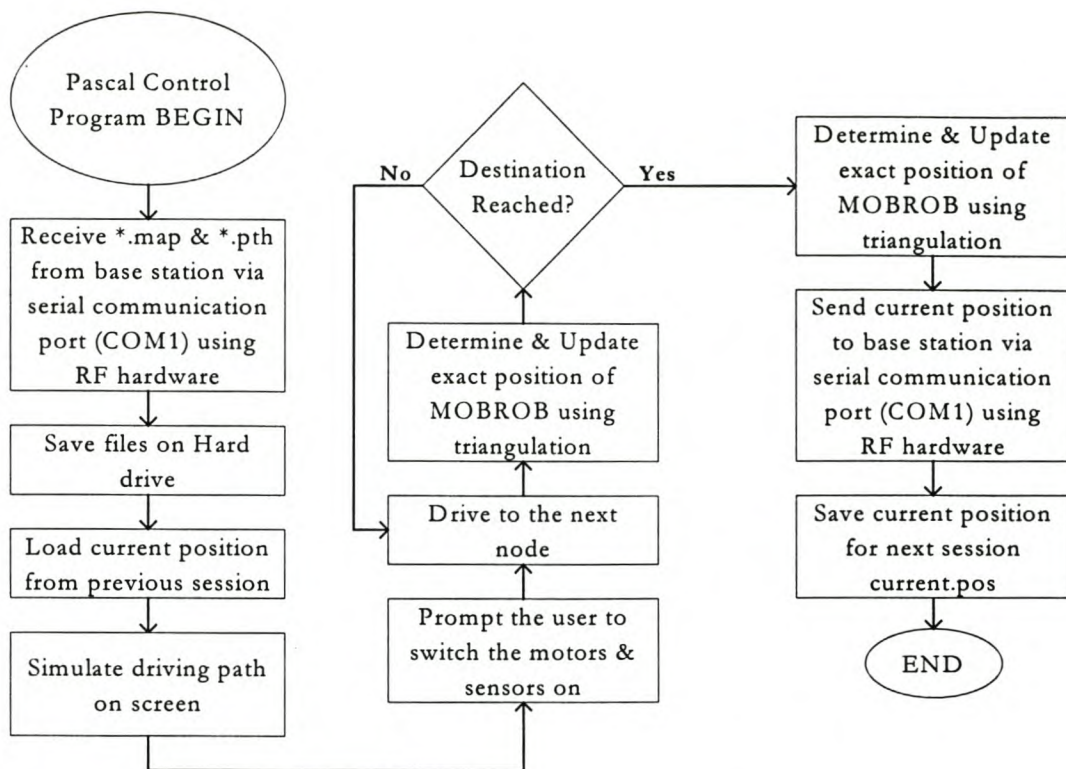


FIGURE 6.5: FLOW DIAGRAM OF PASCAL-BASED PROGRAM

The path MOBROB is going to follow is drawn on the screen. At this stage, the user will be prompted to switch on the motors and sensors. Once this is done, MOBROB continues on the path to the next “turning point” in the path. At this stage the user is asked to enter values for the corners needed to calculate MOBROB’s exact position using triangulation. The hardware for this subset still has to be developed.

MOBROB’s position and its moves to the next “turning point” are updated. Once MOBROB has reached its destination, the triangulation system is used to calculate the exact position. This position is saved in current.pos, ready for the next path. Current.pos is also sent to the base station for path-planning purposes.

6.5 SUMMARY

Flow diagrams for all the main computer programs needed to control MOBROB are given and discussed broadly in this chapter. The flow diagrams are not done in minute detail, but are supposed to guide the reader through the logic of the different programs and not to explain the program code. More detailed descriptions are given in chapter seven.

The program codes are given in Appendices D, E and F. Because of to the absolute enormity of the task; the program code will not be discussed “line-for-line” at all.

7 MOBROB HARDWARE

In this chapter the hardware profile of MOBROB is discussed. Except for a few changes to the RF Units, no changes were made to the hardware profiles of the subsets. The hardware profiles of the subsets are discussed in detail in Chapter Four, therefore this chapter is only a summary and will not include too much detail.

7.1 MOBROB – THE ROBOT

The robot has several interacting parts. The main parts are the following:

- The Vehicle,
- Two Driving Motors,
- The Motion Control Card,
- 24V Onboard Power Supply,
- DC to AC inverter,
- Three Ultrasonic Sensors, and
- Onboard Laptop Computer with Docking Station.

No hardware has ever been developed for the global positioning triangulation subset of the project.

7.2 MOBROB – THE RF UNIT

There are two RF communication units, one for the base station and one for MOBROB. These units are connected to the serial communication ports (COM1) of the two computers. Since the circuits were never documented in the past, circuit diagrams for these circuits are given in

Figure 7.1 and Figure 7.3.

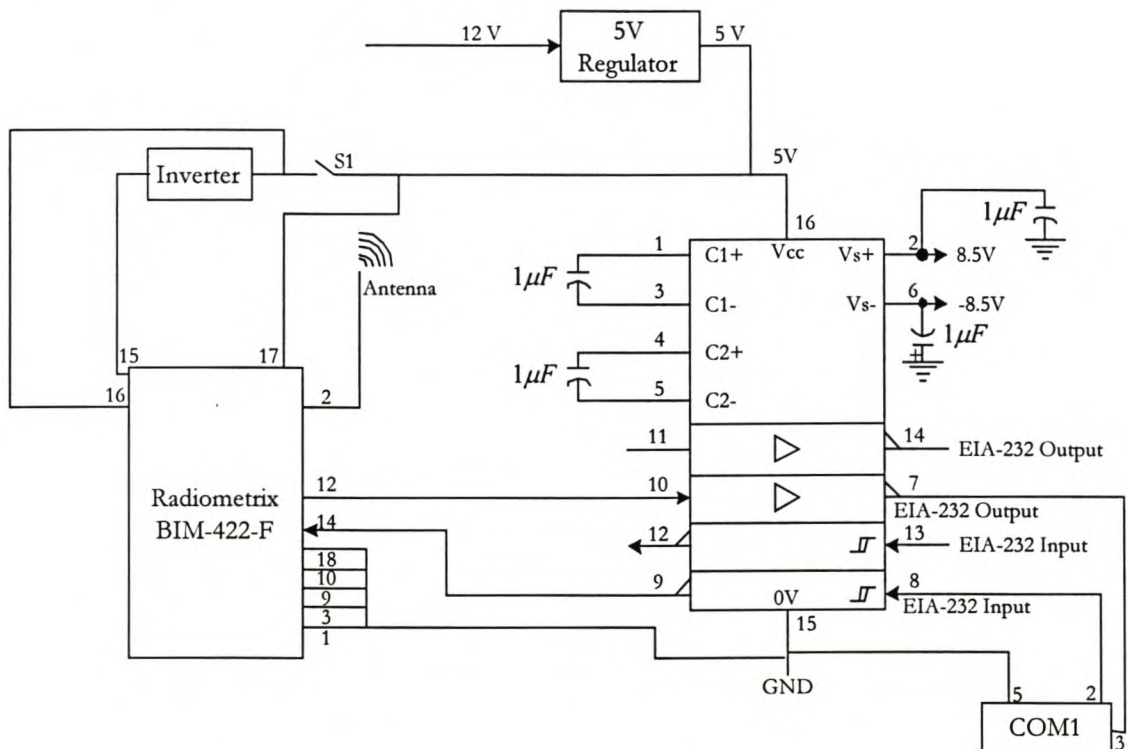


FIGURE 7.1: BASE STATION RF COMMUNICATION UNIT CIRCUIT DIAGRAM

The unit connected to the base station is not automatically switched between the sending and receiving modes of the unit and the user must make this switch. This is done with switch S1 shown in

Figure 7.1. This “normally open” switch must be closed for the unit to be in sending mode. The unit connected to the onboard computer is switched between modes using the parallel port of the computer. For receiving mode, the signal from the parallel port must be 0V. This is shown in Figure 7.3.

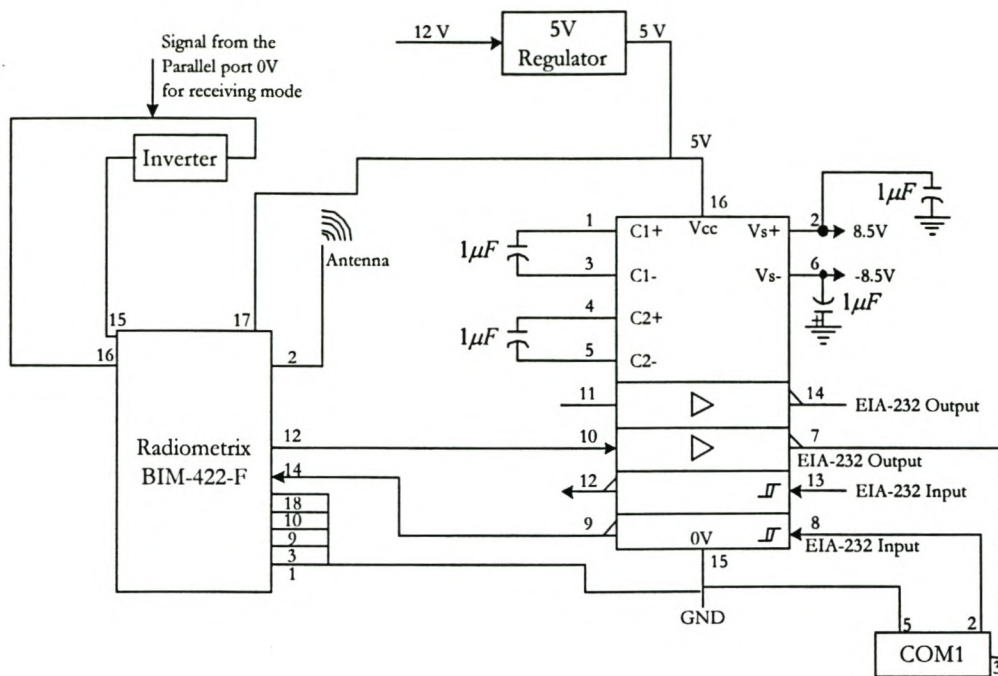


FIGURE 7.3: ONBOARD RF COMMUNICATION UNIT CIRCUIT DIAGRAM

7.3 MOBROB – THE BASE STATION

The base station is a personal computer with the following hardware and software requirements:

1) Hardware:

- Pentium processor,
- 64 Mbytes of RAM, and
- Serial Communication Port (Com1)

2) Software:

- Windows 95, 98 or 2000 operating system,
- MATLAB version 5 or higher and
- All the base station control software for MOBROB.

8 MOBROB SOFTWARE

In this chapter the software programs used for the integrated control system are discussed. Generally, the existing software programs for the subsets were used as a starting point and changed to enable the subsets to work together.

8.1 GENERAL

The software part of the integrated control system was developed in three programming languages, namely:

- BORLAND DELPHI,
- MATLAB, and
- TURBO PASCAL.

Even though this added a great deal to the complexity of the system, each of the languages was used because it has certain abilities that the others do not have.

8.1.1 BORLAND DELPHI'S ABILITIES

BORLAND DELPHI was used to create the main interface on the MOBROB base station. DELPHI programs run in the Windows® environment and can be made user-friendly. It is also possible to write data to the serial communication ports from a DELPHI program. This was required for the RF communication subset of the system.

8.1.2 MATLAB'S ABILITIES

MATLAB's main advantage is that it has the ability to do complex mathematical calculations more effectively than DELPHI. One other important advantage is that MATLAB programs can run in the Windows® environment. MATLAB's greatest disadvantage is that it is difficult to create an executable independent program. Therefore, the user must have MATLAB installed on the base station.

8.1.3 TURBO PASCAL'S ABILITIES

The ease of use and ability to reference hardware directly were the main advantages when using TURBO PASCAL. Because of the fact that TURBO PASCAL is a Dos-based language there is no need for the onboard computer to have a Windows® operating system. In this case, it is an advantage, because the user-friendly interface of Windows® is not required on MOBROB. Thus, the onboard computer may be slower, without influencing MOBROB's performance.

8.2 DELPHI-BASED PROGRAMS

The Delphi-based programs are used as a user interface and to send and receive file via the RF communication hardware. A list of the most important procedures used by these programs, including a short description of the functions of the different procedures, is given in Table 8.1.

TABLE 8.1: A LIST OF THE DELPHI PROCEDURES WITH A DESCRIPTION OF EACH

Procedure name	Description
TfrmMain.About1Click	Shows the about-form as a modal.
TfrmMain.PathPlanningClick	Opens MATLAB and the path-planning program.
TfrmMain.QPathPlanningClick	Opens MATLAB and the quick path-planning program.
TfrmMain.ButtonQTransmitClick	Reminds the user to switch on the RF communication software, sends the last quick-planned path wmap.pth and wmap.map to MOBROB, reminds the user to switch the RF communication hardware to receiving mode and receives the current.pos file..
TfrmMain.ButtonTransmitClick	Reminds the user to switch on the RF communication software, sends a user-selected path to MOBROB, reminds the user to switch the RF communication hardware to receiving mode and receives the current.pos file.
TfrmMain.FormCreate	Sets the serial communication port, COM1's parameters when the form is created

Procedure name	Description
	when the form is created.
TRemSend.OKButClick	Returns from the reminder, reminding the user to switch the RF communication hardware to sending mode to the main form.
TRemReceive.OKButClick	Returns from the reminder reminding the user to switch the RF communication hardware to receiving mode to the main form.

8.3 MATLAB-BASED PROGRAMS

The MATLAB-based programs are responsible for the path-planning of MOBROB. In Table 8.2 a list of all the procedures used in the process of path planning is given. A few screenshots of certain parts of the path-planning process follow the table.

TABLE 8.2: A LIST OF THE MATLAB PROCEDURES WITH A DESCRIPTION

Procedure name	Description
Main	This is the main procedure of the Path-planning program. This procedure creates the user interface (Figure 8.1) and calls up the other procedures when needed.
Mainqp	This is the main procedure of the Quick Path-planning program. It creates the user interface and calls up the other procedures when needed.
LoadMap	This procedure loads and draws on the screen the MATLAB map file the user selected (Figure 8.3) from the open file dialogue box. The potential field file is also loaded for future use.
LoadMapqp	This procedure loads and draws on the screen the MATLAB map file wmap.mop. The potential field file wmap.pot is also loaded for future

Procedure name	Description
	use.
CreatMap	This procedure creates a new MATLAB map file and a new potential field file. The user defines the dimensions of the room (Figure 8.2) and creates obstacles inside the room.
EditMap	The user opens an existing MATLAB map file and adds or removes obstacles from the file. The new potential field file is created and saved.
LoadPos	The current position of MOBROB is loaded for future use.
Starget	This procedure gets MOBROB's destination from the user and calls up the walk procedure.
Stargetqp	The quick path-planning program uses this procedure. The user defines MOBROB's destination. The procedure then calls the walkqp procedure.
Walk	This procedure controls the path-planning operation. Posib and Cream are called from here and the decoding of the path to create the *.map and *.pth (Figure 8.4) files is done here.
Walkqp	The path-planning operation for the quick path-planning program is done here. Posib and Cream are called from here and the decoding of the path to create the wmap.map and wmap.pth files is done here.
Posib	The " <i>potential fields</i> " of the three blocks MOBROB can drive to are determined. The three blocks are the one directly ahead of MOBROB and that block's two neighbours. The actual potential fields as calculated by potmap2 are evaluated two levels deep to make the best selection.
NewPos	Once the three potential fields of the three options of MOBROB's current position are determined, the lowest one of the three is selected and the robot's new position is logged.
PotMap	Calculates the potential field values for the potential field file.

Procedure name	Description
PotMap2	Calculates the potential field values once the destination is known.
Cream	Cream smooths the path logged so far. The result of Cream is a number of co-ordinates where MOBROB must change its direction and drive to the next set of co-ordinates in the path.

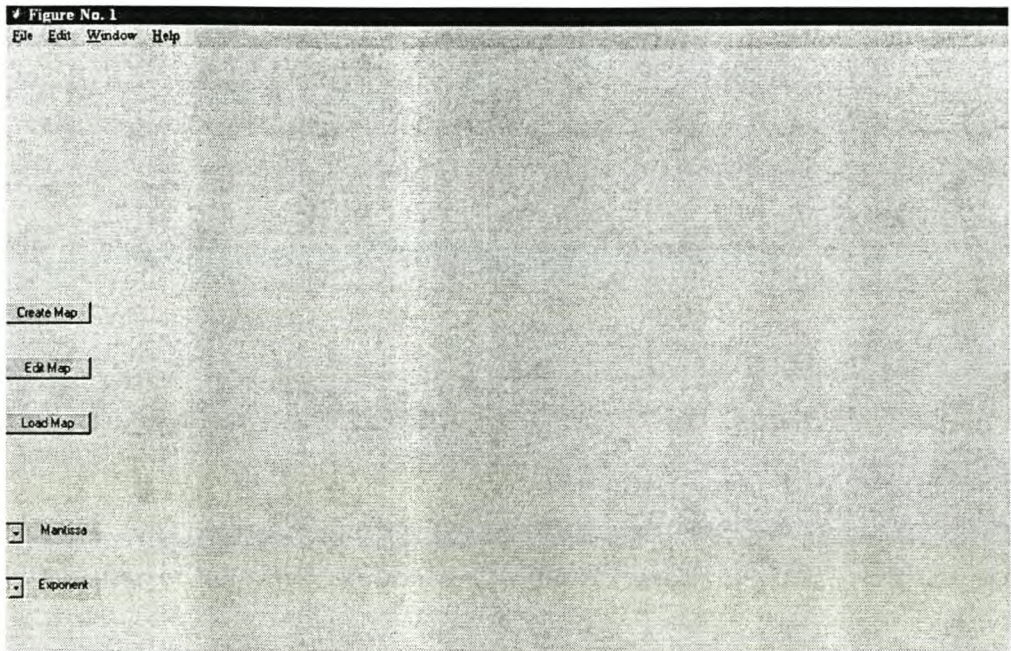


FIGURE 8.1: USER INTERFACE CREATED BY THE MAIN PROCEDURE

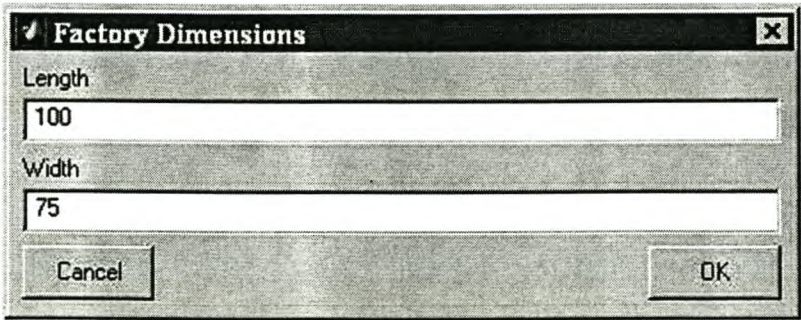


FIGURE 8.2: THE USER DEFINES THE ROOM DIMENSIONS

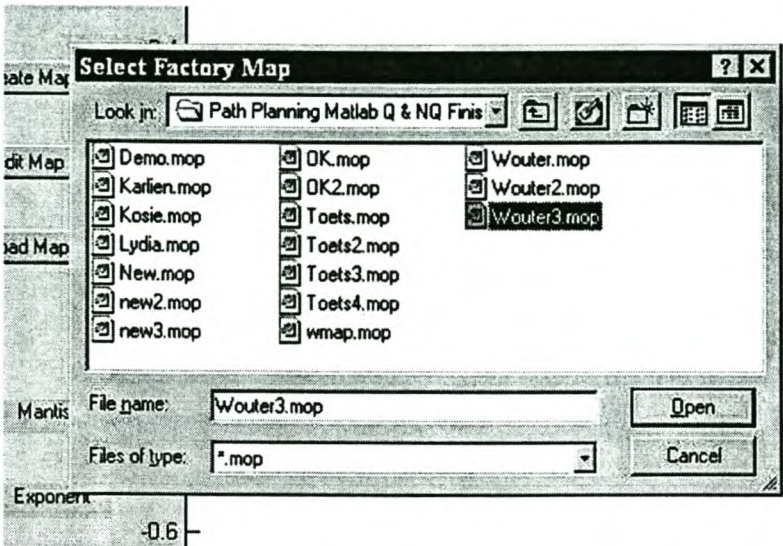


FIGURE 8.3: SELECT A MATLAB MAP FILE TO LOAD

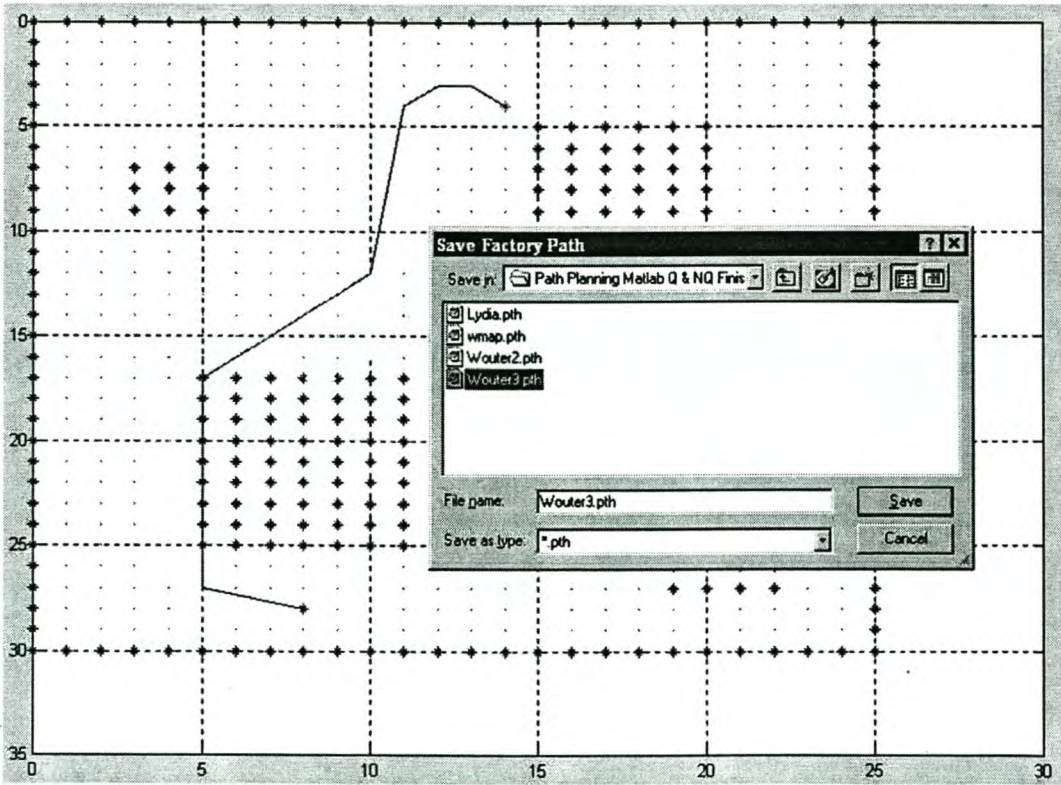


FIGURE 8.4: THE USER ENTERS A FILENAME TO SAVE THE CREATED PATH

8.4 PASCAL-BASED PROGRAMS

The PASCAL-based programs are responsible for the control of the robot itself. They are the only programs on the onboard computer. In Table 8.3 a list of the procedures used in the main control program is given. The author did not change any of the units developed earlier by Nel (1997) or Steenkamp (1998) and these units are therefore not included in this list. For more information on the units used in this main control program, see Nel's (1997) document on The Design and Construction of a Mobile Robot for Material Handling and Steenkamp's (1998) document on Communication with a Mobile Robot using Radio Frequencies.

TABLE 8.3: A LIST OF THE PASCAL PROCEDURES WITH A DESCRIPTION

Procedure name	Description
EmergencyStop	Executes an emergency stop when the emergency stop button is selected.
SetupRTGraph	Creates the graphics on the screen of the onboard computer.
UpdateDisplay	Updates the graphics on the screen from time to time.
GetGoalSteerAngle	Determines the direction MOBROB must head in order to drive to the next node.
AdjustHeadAngle	Changes the format of the direction MOBROB is facing to simplify future calculations.
GetNodePath	Reads the path file received from the base station for future use.
SimulateDriveToNextNode	Draws the path to the next node on the screen.
Simulator	Simulates the path MOBROB will follow.
WaitForMoveEnd	Waits for MOBROB to reach the next node before the program is continued.
DriveToNextNode	Controls the driving motors so that MOBROB drives to the next node.
ControlMOBROB	Controls the whole journey, until MOBROB reaches its destination.

Procedure name	Description
OpenWindows	Opens the “ <i>windows</i> ” used on the screen.
CloseWindows	Closes the “ <i>windows</i> ” used on the screen.
ReceivFiles	Receives the files sent from the base station and saves them in the correct directory.
CreatePos	Creates the file <code>current.pos</code> .
SendFile	Sends the <code>current.pos</code> file to the base station.
UpdatePosition	Calculates the exact position of MOBROB, using the global positioning system and updates all position parameters.

8.5 DIFFERENT FILE FORMATS

The aim of this project was to integrate the several subsets of the MOBROB project. The MATLAB-based path-planning subset and the PASCAL-based control subsets both need text files for their inputs and both create text files as their outputs. However, the file formats of these files are not the same.

The first large difference is that the axis of the MATLAB programs and Pascal programs are opposite. The positive x-axis in the MATLAB programs is the positive y-axis in the PASCAL program and the positive x-axis in the PASCAL program is the positive x-axis in the MATLAB programs. The PASCAL-based control program running on MOBROB needs two files before it is able to drive to its destination. The extensions of these files are `.map` and `.pth`. The file formats are shown in Figure 8.6 (map file) and Figure 8.7 (path file). In these files, the axis was already swapped. The `main.m` and `mainqp.m` MATLAB programs did this. In the original control program, the map was used to represent a room or factory floor map where the nodes indicated stopping points on the floor. Every node was connected to other nodes and the factory floor was represented through a weakly-connected graph. A new map file is now created for every path and the nodes are the co-ordinates where the robot must stop. Node zero is the current position and the first leg of the path is from node zero to node one. The path file (Figure 8.7) was originally used to give the nodes visited in the path and the order of visiting. In theory, this is still exactly the same, but MOBROB will always be at node

zero, visiting the nodes in order until it reaches node x , where x represents the number of nodes in the map file.

The advantage of this system is that it is possible for the user to select any “legal” destination on the factory floor and the control system will create a path to that destination. A *legal* destination is a destination that is not on an obstacle. Thus, the nodes do not bind MOBROB and the robot is far freer-ranging than before. The previous system was bound with a maximum of twelve nodes per map and it was necessary to change the maps often to incorporate different destinations. With the new system, only one user-defined map is needed, the MATLAB map (*.map) shown in Figure 8.8, and any destination on this map can be reached. The *.map, *.pth and MATLAB potential field file (.pot) shown in Figure 8.9 are created automatically.

In Figure 8.10 the format of the current.pos file is shown. This file is used to save the current position of the robot between trips. It is created by the PASCAL-based control program at the end of a journey and used by the MATLAB-based path-planning program to plan the next trip. The first number in the file is the x co-ordinate and the second number is the y co-ordinate of MOBROB’s current position for the MATLAB-based programs. The third number is an integer between 1 and 8 and represents the current direction of the robot (facing direction). The scale used to determine this integer is given in Figure 8.5. In the MATLAB path-planning programs, the direction 1 is in the direction of the negative y -axis and the direction 3 is in the direction of the positive x -axis. For the PASCAL control program on MOBROB, 0° is in the direction of the positive y -axis and 90° is in the direction of the positive x -axis. This is graphically shown in Figure 8.5. In this figure, positive x and positive y are indicated as used by PASCAL.

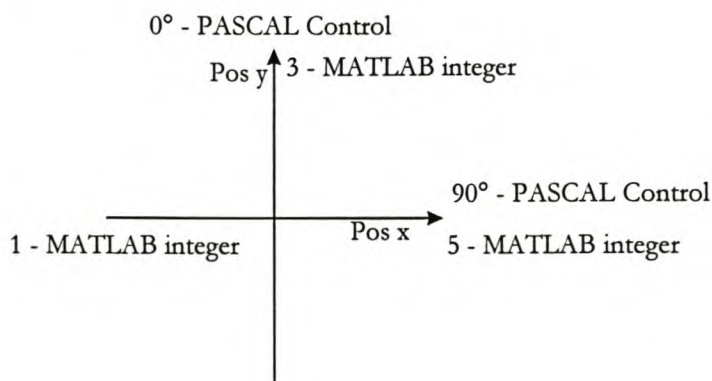


FIGURE 8.5: DIFFERENT DIRECTION SYSTEMS USED BY MATLAB PATH-PLANNING AND PASCAL CONTROL

TABLE 8.4: SCALE USED TO DETERMINE THE FACING DIRECTION OF MOBROB

Facing direction in Current.pos	Facing direction in degrees, where 0° is in the direction of the positive y-axis, 90°
1	$-112.5^{\circ} \leq x < -67.5^{\circ}$
2	$-67.5^{\circ} \leq x < -22.5^{\circ}$
3	$-22.5^{\circ} \leq x < 0^{\circ}$ & $0^{\circ} \leq x < 22.5^{\circ}$
4	$22.5^{\circ} < x \leq 67.5^{\circ}$
5	$67.5^{\circ} < x \leq 112.5^{\circ}$
6	$112.5^{\circ} < x \leq 157.5^{\circ}$
7	$-180^{\circ} < x \leq -157.5^{\circ}$ & $157.5^{\circ} < x < 180^{\circ}$
8	$-157.5^{\circ} \leq x < -112.5^{\circ}$

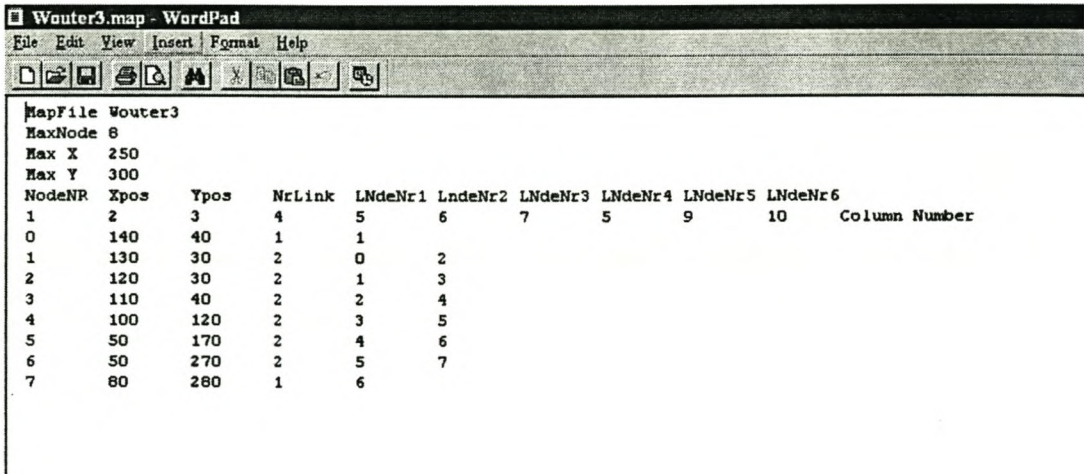


FIGURE 8.6: *.MAP FILE USED BY THE PASCAL-BASED CONTROL PROGRAM

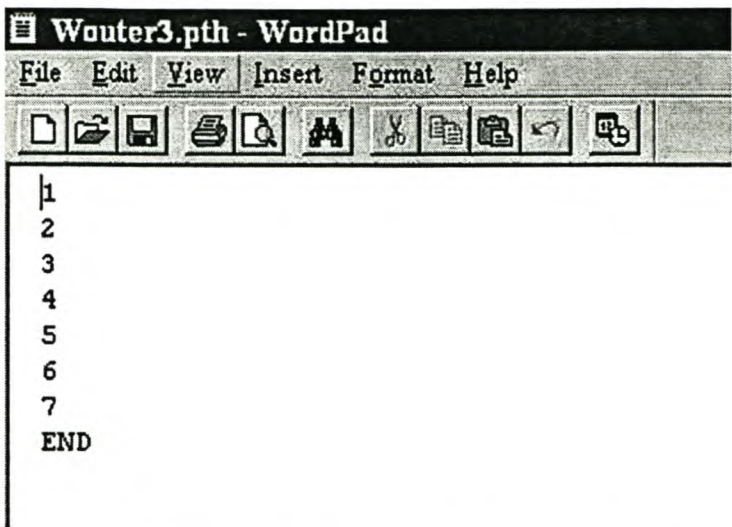


FIGURE 8.7: *.PTH FILE USED BY THE PASCAL-BASED CONTROL PROGRAM

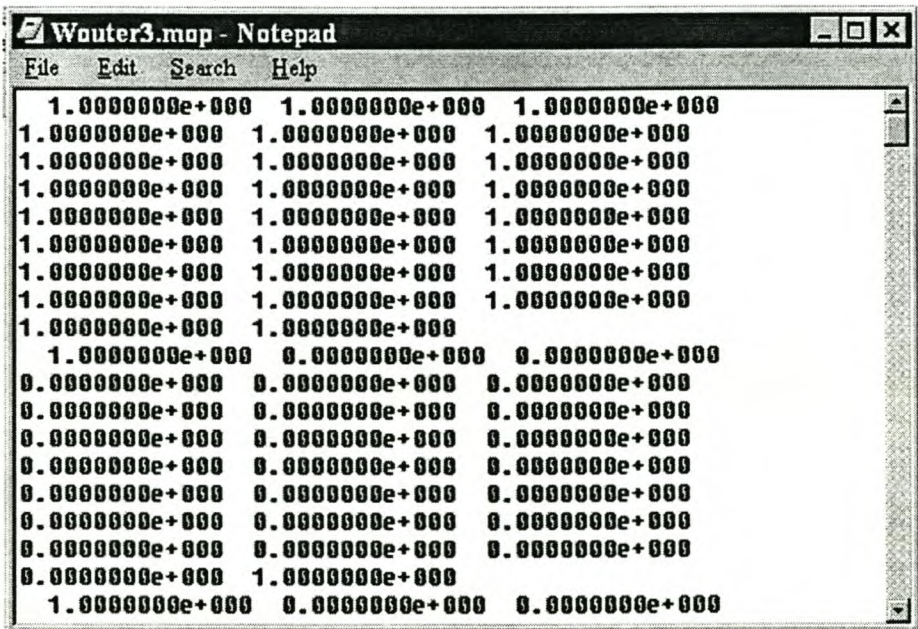


FIGURE 8.8: MATLAB MAP FILE

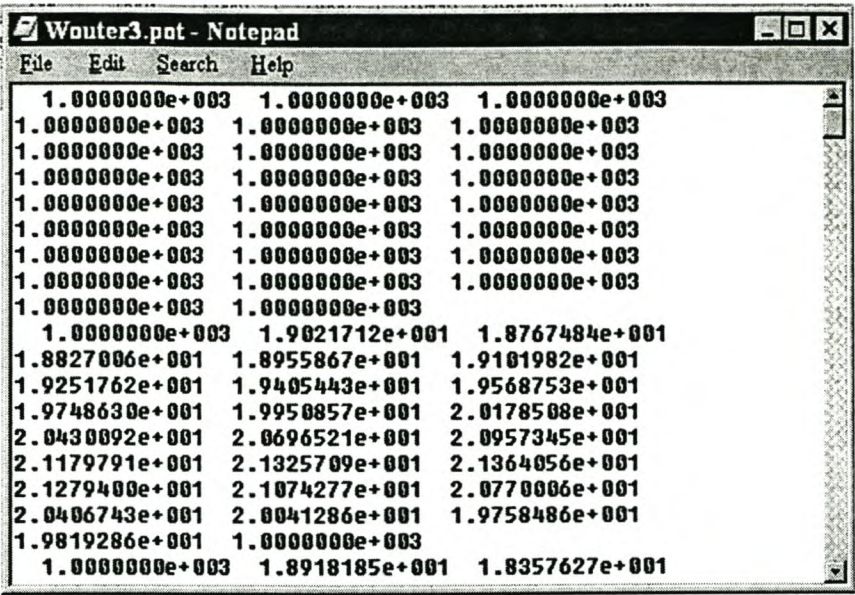


FIGURE 8.9: MATLAB POTENTIAL FIELD FILE

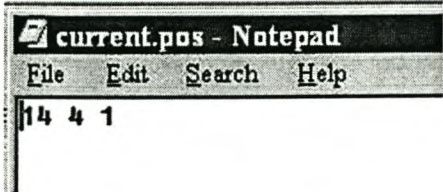


FIGURE 8.10: CURRENT.POS FILE USED BY BOTH THE CONTROL PROGRAM ON MOBROB AND THE MATLAB PATH-PLANNING PROGRAM

9 CONTROL SYSTEM CONCLUSIONS

9.1 GENERAL

The author was successful in debugging, rewriting, redesigning and rebuilding the necessary parts of the existing subsets.

Where applicable, this was done taking integration into account to ease the actual integration of the subsets. The two subsets that required the most work was the path planning subset and the RF communication subset.

The path planning software had to be rewritten to a great extent. Some alterations were necessary as well to render the solution more practical.

The RF communication hardware was not in working order and was redesigned and rebuild completely. The RF communication software was also altered to make integration easier. No RF communication software for DELPHI existed and this software was written, taking integration into account.

9.2 THE INTEGRATED CONTROL SYSTEM

The author created an integrated system that can be used to control the movement of MOBROB. This system is software based and consists of three separate, but integrated parts.

The two subsets most unfriendly to integration with the vehicle and the onboard control of the vehicle were the path planning subset and the RF communication subset. The author found that it was easier to rewrite the software of these subsets, taking integration into account, than to integrate the existing software. Since the author had to redo large parts of these subsets this was possible.

9.3 THE MOBROB HARDWARE

The only MOBROB hardware that needed alterations was the RF communication hardware. These two units were entirely redesigned and rebuild.

No hardware exists for the global navigation system. The accuracy of the angles required for this subset must

be very high for the subset to be accurate. The hardware recommended in the original project is not very practical and needs some rethinking.

9.4 THE MOBROB SOFTWARE

The MOBROB software is integrated and in working order. It is difficult to put a boundary around the integrated control system, because it is difficult to determine where the subsets end and where integration starts. The result, however is an integrated system.

SECTION FOUR: VALIDATION

This section represents the completion of the project. The results obtained are discussed, conclusions are reached and recommendations are made.

10 DISCUSSION OF RESULTS

This chapter discusses the results by “walking through” the whole process once. This walkthrough is mostly illustrated through screenshots.

10.1 STATION CONTROL

The station control program gives the user four options. This is illustrated in Figure 10.1.

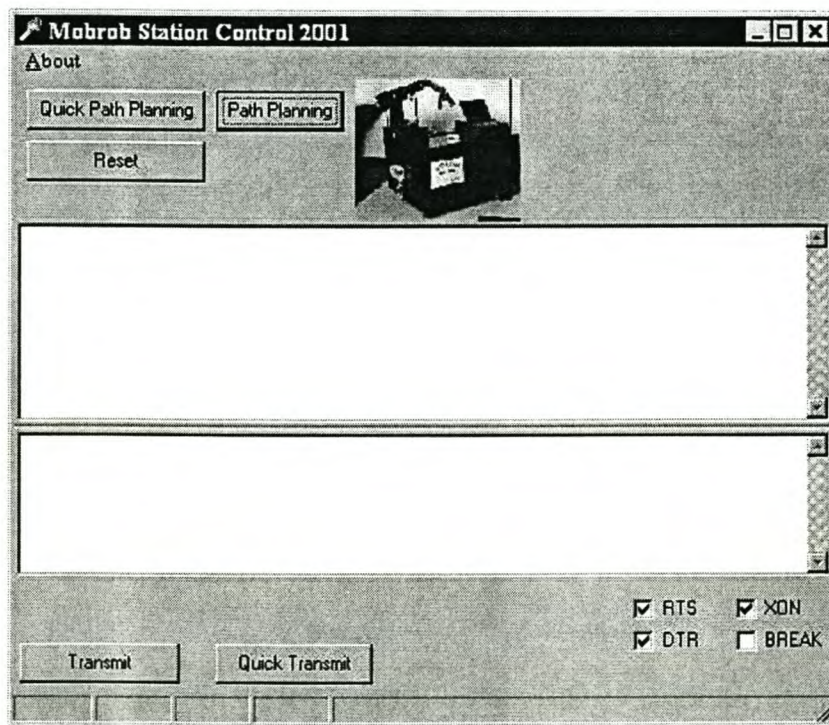


FIGURE 10.1: STATION CONTROL PROGRAM

10.2 CREATING A NEW MAP

The first step in the process is to create a new map for the factory floor. Path-Planning must be selected in the station control program. This will open MATLAB and create map can be selected. A map was created for the room currently used by SENROB at the University of Stellenbosch. In Figure 10.2 the floor dimensions have been entered. A new map is created when the OK button is pushed. Stars that represent the walls of the room

surround the map. This can be seen in Figure 10.3. No obstacles have been added to the map at this stage.

The room has only one large obstacle in the centre. The outlines of this obstacle were added before the map was saved (Figure 10.4).

At this stage the map was saved (Figure 10.5). The map will be finished using the Edit Map procedure.

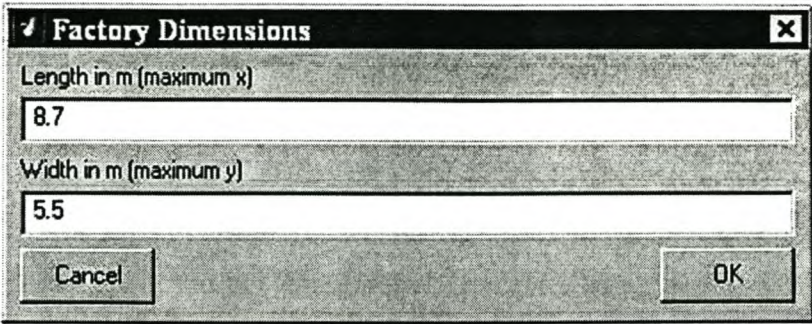


FIGURE 10.2: USER DEFINED DIMENSIONS

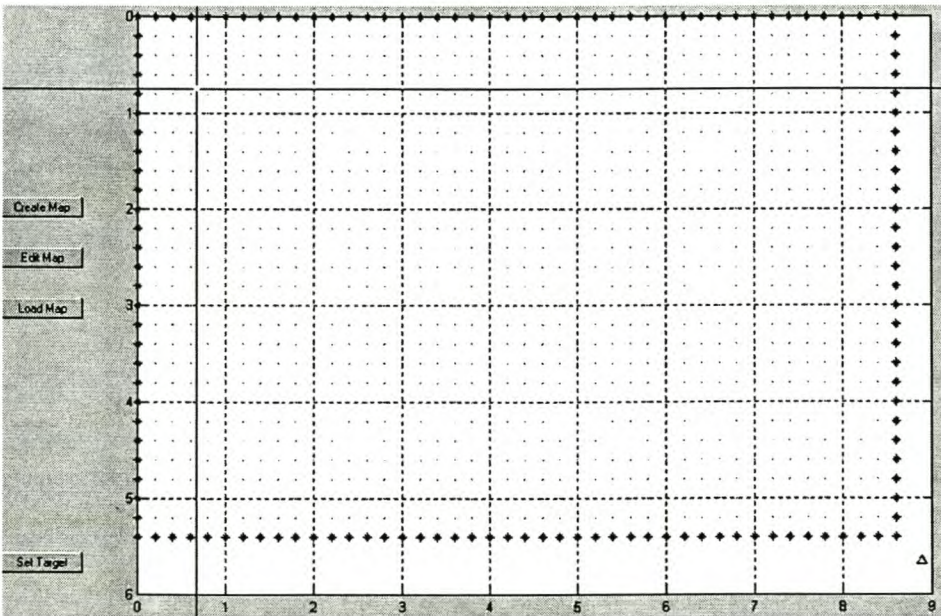


FIGURE 10.3: NEW FACTORY FLOOR WITHOUT ANY OBSTACLES

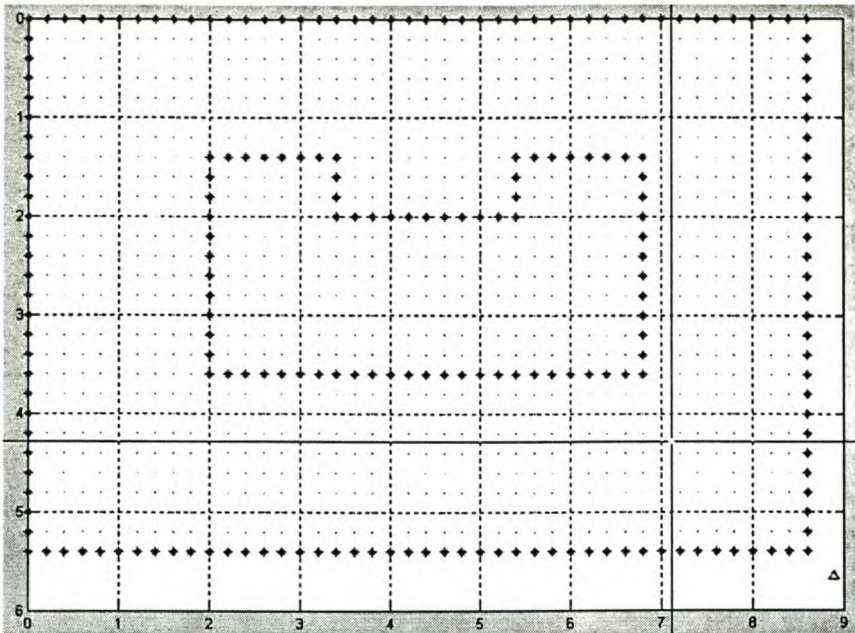


FIGURE 10.4: OUTLINE OF THE OBSTACLE IN THE ROOM ADDED

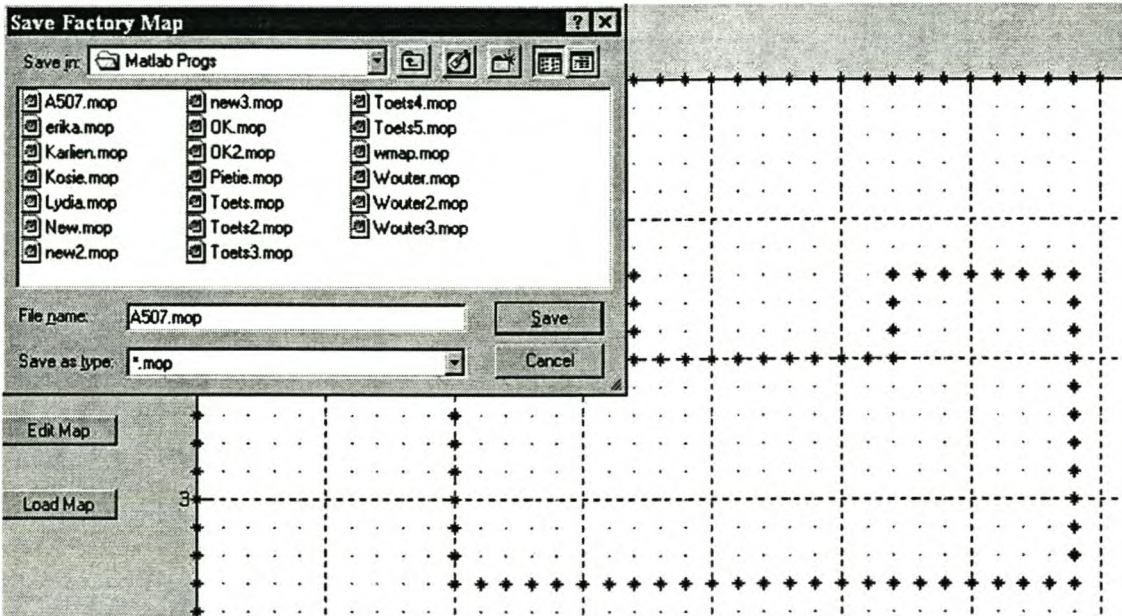


FIGURE 10.5: SAVING FACTORY FLOOR MAP FOR FUTURE USE

10.3 EDITING AN EXISTING MAP

The map created in the previous paragraph will be opened (Figure 10.6) and finished using the Edit Map procedure in this paragraph. In Figure 10.7 the inside of the obstacle is being filled. The obstacle has not grown yet. The finished map with the grown obstacle and walls is shown in Figure 10.8.

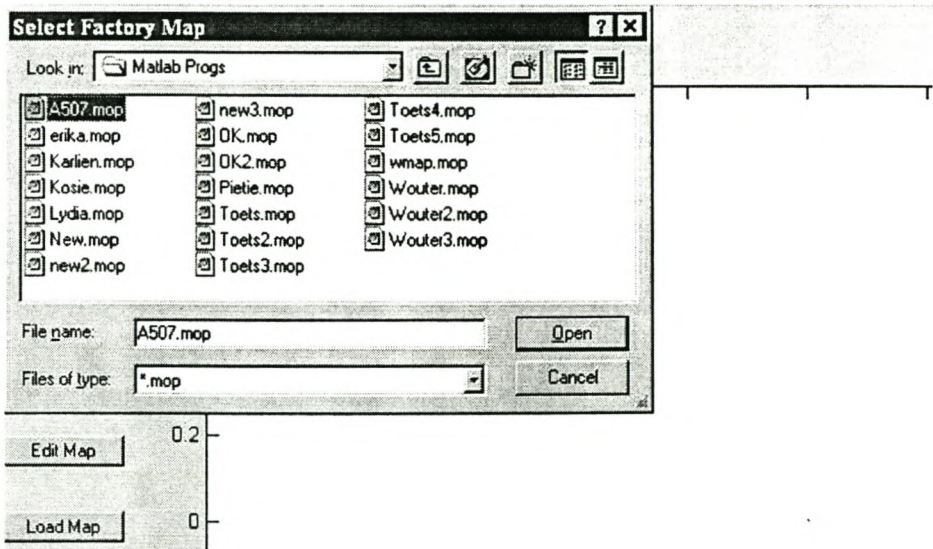


FIGURE 10.6: OPENING A MAP FOR EDITING

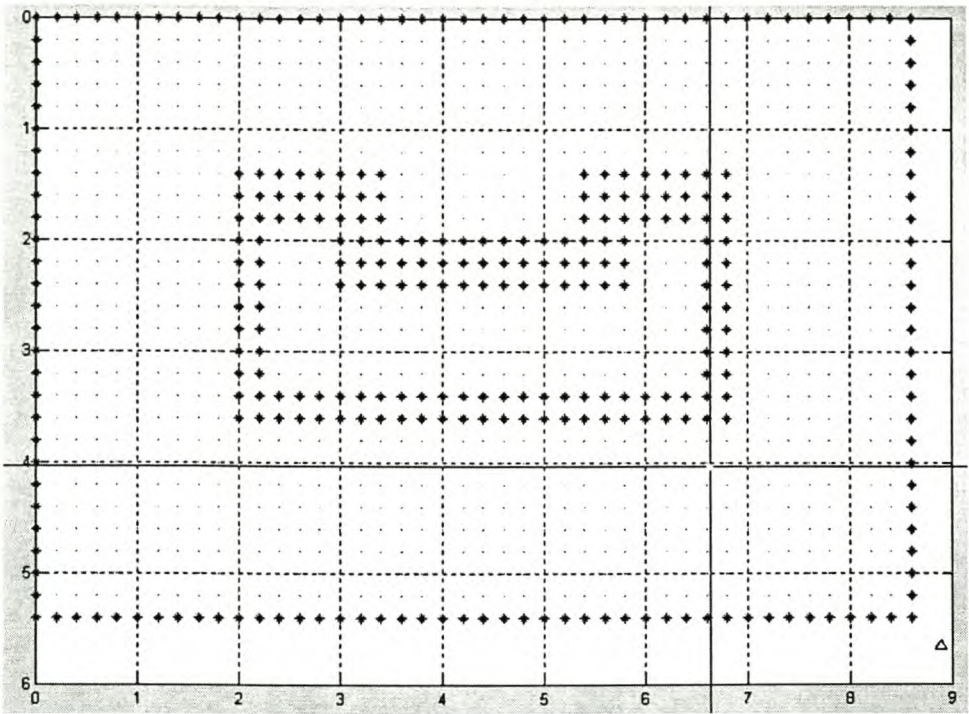


FIGURE 10.7: FILLING THE INSIDE OF THE OBSTACLE

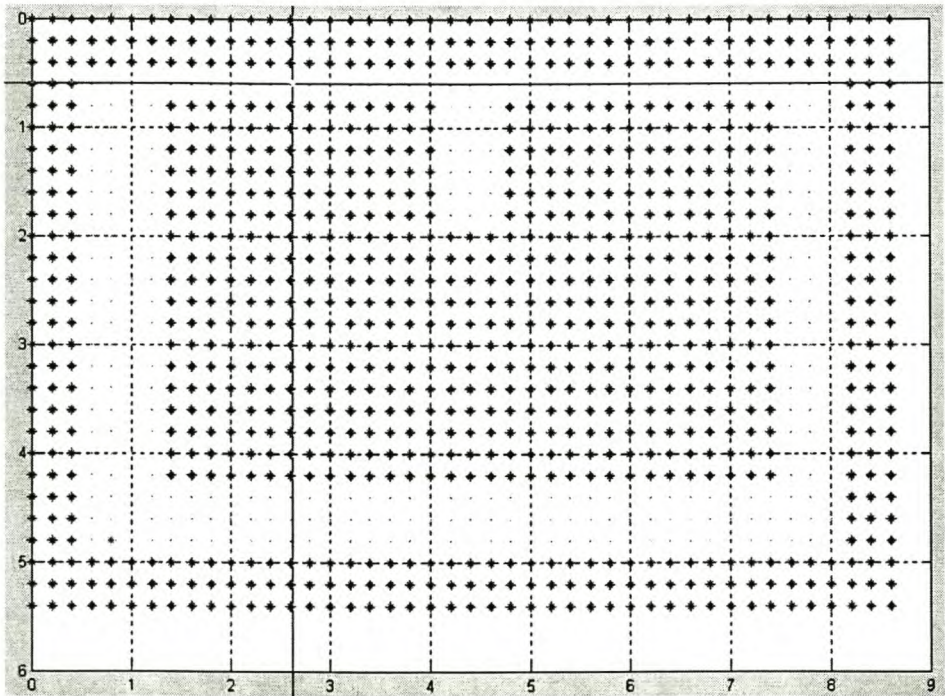


FIGURE 10.8: FINISHED MAP WITH GROWN OBSTACLE AND WALLS

10.4 PATH-PLANNING

Once the map is completed, a path can be planned. Since the quick path-planning files for the new map have not been created yet, the first path planned for a new map must be done using the path planning program and not the quick path planning program. The current.pos file must also be edited the first time a new map is used. This can be done in Notepad.

After the current.pos file has been updated, the map may be loaded. The map with the star showing MOBROB's current position can be seen in Figure 10.9. A destination is selected and a path created. This new planned path is given in Figure 10.10.

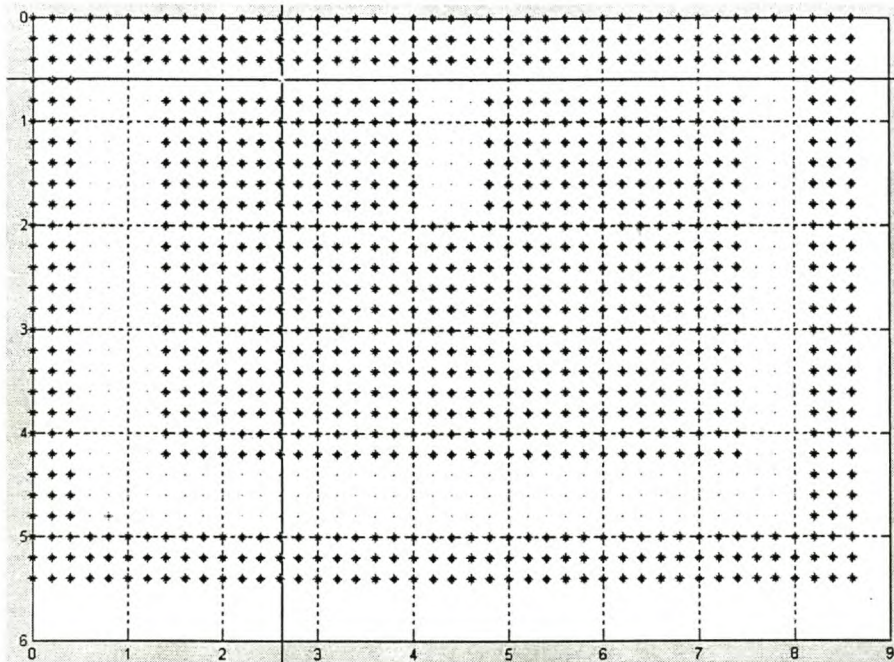


FIGURE 10.9: LOAD MAP FOR PATH-PLANING

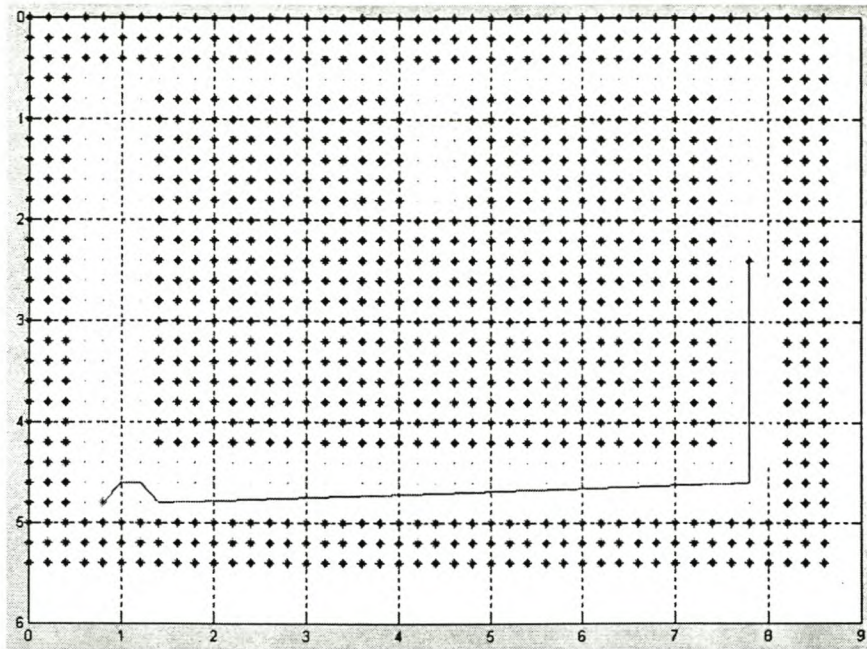


FIGURE 10.10: PATH CREATED BY MATLAB

10.5 QUICK PATH-PLANNING

To execute the quick path-planning program, the user must return to the Station Control program and select Quick Path-Planning. The map created in the previous paragraphs is opened, with MOBROB's current position. A path is planned with the quick path-planning program. This path is shown in Figure 10.11.

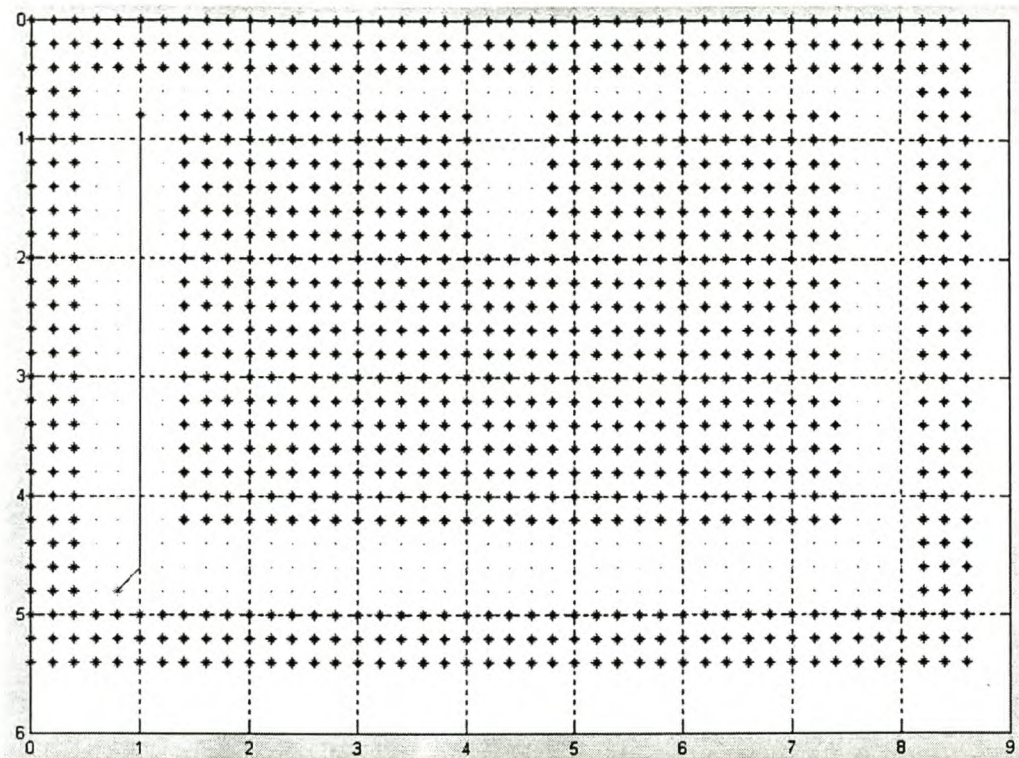


FIGURE 10.11: PATH PLANNED USING THE QUICK PATH-PLANNING PROGRAM

10.6 SEND A PATH TO MOBROB

Select Transmit in the Station Control Program. An Open dialog box will appear. Select the *.map file that must be sent to MOBROB (Figure 10.12). The user is reminded to switch on the RF communication hardware and to switch it to sending mode. Once the file has been sent, the user is again reminded to switch the RF communication hardware back to receiving mode (Figure 10.13).

10.7 SEND A QUICK-PLANNED PATH TO MOBROB

Select the Quick Transmit button. The last quick-planned path will be sent to MOBROB.

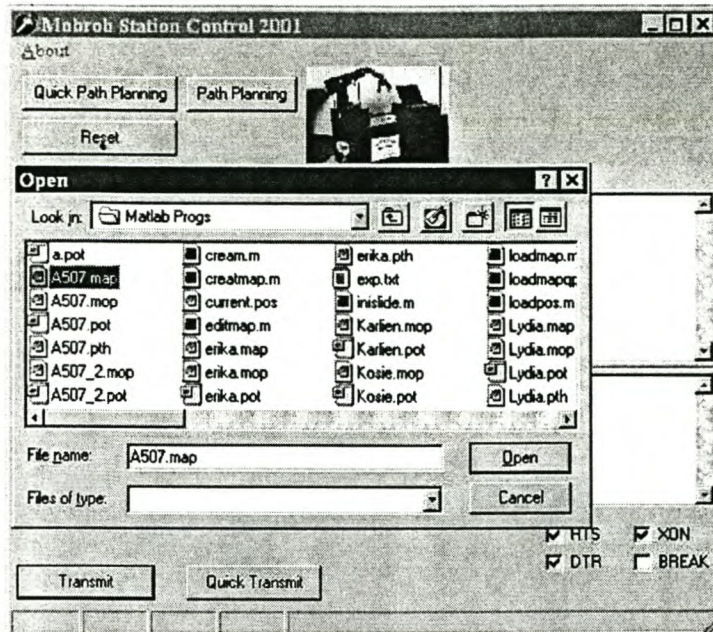


FIGURE 10.12: SEND SELECTED FILE TO MOBROB

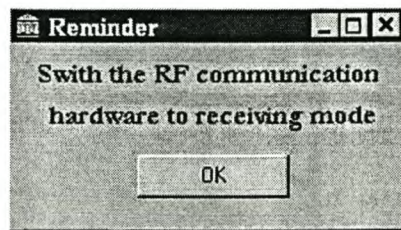


FIGURE 10.13: THE USER IS REMINDED TO SWITCH THE RF COMMUNICATION HARDWARE BACK TO RECEIVING MODE

10.8 MOBROB DRIVES TO DESTINATION

Once MOBROB receives the files it can draw the path on the screen (Figure 10.14) and drive to its destination (Figure 10.15). The `current.pos` file (Figure 10.16) is updated and then it is sent back to the base station.

This is the last part of the walkthrough. The whole process has been completed and can start over again.

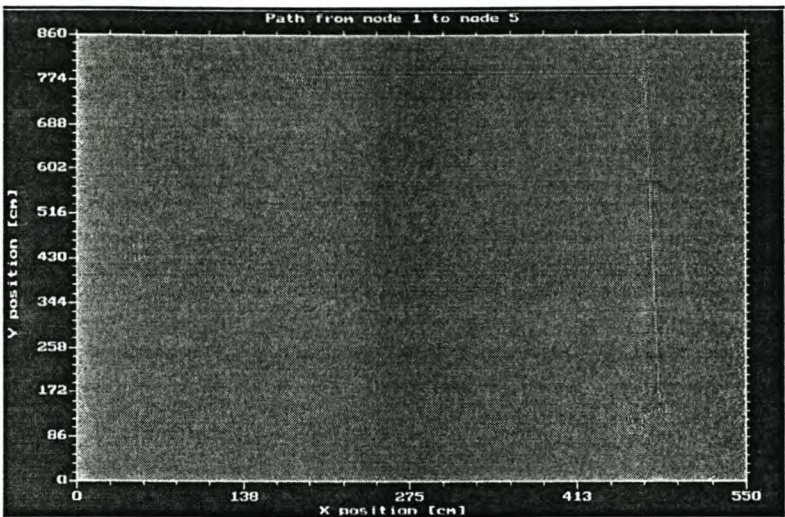


FIGURE 10.14: PATH ON THE ONBOARD COMPUTER

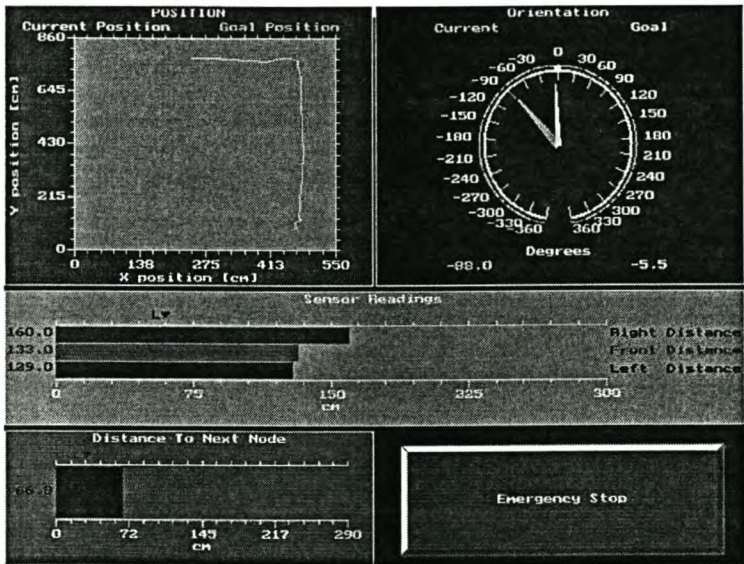


FIGURE 10.15: MOBROB DRIVING TO ITS DESTINATION

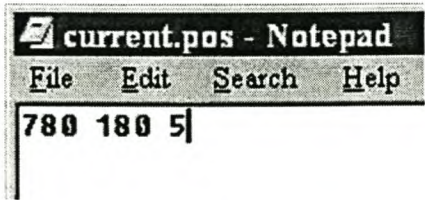


FIGURE 10.16: NEW CURRENT.POS FILE

10.9 EVALUATION OF RESULTS

This section evaluates the results obtained during this project, by listing advantages and disadvantages of the integrated control system.

10.9.1 *ADVANTAGES OF THE SYSTEM*

The implemented integrated control system has the following advantages:

- The system successfully integrates all of the previously unintegrated subsets of the MOBROB project.
- The user interface on the base station is user friendly and written with a Windows® interface.
- It is easy to plan a new path and MOBROB's position is automatically updated after each movement.
- The programs running on the onboard computer is written in DOS-based TURBO PASCAL and there is no need for this computer to have a Windows operating system and an slower and therefore less expensive computer may be used. The user-friendly interface required on the base station is not required here.
- Computer hardware is used more effectively than before. The parallel port on the base station is not used and is available for other applications.

10.9.2 *DISADVANTAGES OF THE SYSTEM*

The disadvantages of the system are as follow:

- The base station must have MATLAB installed and this package is expensive.
- The factory floor is divided into 20x20 cm blocks. This grid may not be fine enough for docking. This grid size was chosen to shorten the computer time needed for path-planning.
- It is possible for MOBROB to drive to a position where it cannot drive out again, since it cannot perform U-turns.

11 FINAL CONCLUSIONS

On completion of this project, certain conclusions were made. These conclusions are discussed briefly in this chapter.

11.1 GENERAL

The author found this project very interesting, and complex. The integration of subsets unfriendly to integration needs both perseverance and creativity.

11.2 COMPUTER INTEGRATED MANUFACTURING

Computer-Integrated Manufacturing or CIM includes the integration of the whole manufacturing enterprise and all its parts. Everybody and every system in the enterprise must be involved in this process, otherwise it is unlikely that it will be successful.

The complexity of systems increases very rapidly when different systems are integrated and therefore the complexity of each individual system must be minimised.

11.3 THE EXISTING MOBROB PROJECT

The existing MOBROB projects as it was found in the AS-IS state at the beginning of the project is a perfect example of a system where integration was not taken into account when the different subsets were designed and completed. This may be due to the main projects running concurrently or the lack of experience of the people working on the different subsets.

Certain subsets were not in working order, program codes needed some debugging and were incomplete and some of the hardware was not functioning as documented.

Certain hardware parts for the subsets still need to be developed. Hardware for the global navigation system is one example of this.

Lack of communication between the different developers is identified as one of the main reason for the

difficulty of integration. The other two main reasons are time and people. At the beginning of the project the author saw these two as the two main reasons, but have come to believe otherwise while completing the project.

11.4 THE INTEGRATED CONTROL SYSTEM

For some of the subsets, it was easier to alter the subset than to integrate the subset as it was. None of the theory behind the subsets was changed. This was seen as a given by the author.

The author concludes that the integrated control system proposed in this document is one solution to a complex problem. It is by no means the only solution to this problem.

Even though the system was not tested against other systems, the author believes that the solution is working properly and that the objectives of this project were met.

11.5 FINAL REMARKS

The last and most important conclusion made after this project is that it is extremely important to integrate subsets from the concept phase of a project and to continue with this process throughout the designing and finishing phase.

Communication is essential in any project and if effective communication exists between the developers of a system, many of the time-consuming problems will be solved faster and more effectively.

12 RECOMMENDATIONS

The author gained valuable experience during this project and will make a few recommendations for future projects.

12.1 GENERAL

When new hardware is developed for subsets that has no hardware, it is recommended that it should be integrated with the system immediately.

The same is recommended for any subsets added to the system. A new subset should not be tackled before the previous one has been integrated and the whole system is working together perfectly.

12.2 CONCURRENT INTEGRATION

If concurrent integration is applied and the subsets are friendly to integration, the final integrating of the subsets will not only be much easier, but money and time will be saved.

It is also recommended that researchers are encouraged to communicate with each other when parts of a project are developed simultaneously. One of the researchers should be assigned as project leader and short meetings from time to time will make this easier.

Projects should be documented if future work on the project is considered. Program codes should be complete, final and available in electronic format.

12.3 MOBROB – THE ROBOT

If SENROB considers building a new mobile robot in the future, it is recommended that the shape of the robot chassis be circular. This will make the robot more manoeuvrable, improve obstacle avoidance and simplify docking.

13 SUMMARY

The MOBROB project at the Centre for Robotics (SENROB), Industrial Engineering Department, Stellenbosch University started several years ago with students related to the Industrial Engineering Department developing subsets for a mobile robotic system as final year projects and masters theses. This mobile robotic system is referred to as MOBROB, which is an acronym for MOBile ROBot.

The subsets developed were:

- Fuzzy logic controller for an automated guided vehicle (Deist, 1993),
- Design and construction of a mobile robot for material handling (Nel, 1997),
- The mapping and path-planning of an industrial mobile robot (Van Rooyen, 1997),
- A Study on the different navigational techniques for a mobile robot (Mentz, 1997), and
- Communication with a mobile robot using radio frequencies (Steenkamp, 1998).

Due to the fact that time and people separated the subsets, they turned out to be unfriendly to integration. All the subsets are very important to the MOBROB project and the proper functioning of the system, but none of them are very useful as loose standing subsets, functioning without the other.

Therefore, integration was necessary to create a system where all the subsets are allowed to contribute to the system. This integrated system or control system was the main objective of this project. The successful completion of the other objectives listed in chapter one was necessary in order to equip the author with the necessary expertise to complete this main objective successfully.

13.1 LITERATURE REVIEW

The literature review consisted of three parts or chapters. The first chapter is a general introduction to computer integrated manufacturing (CIM). The aim of this chapter was to highlight the need of integration and to illustrate that integration becomes a time consuming and expensive exercise if it is not taken into account from the beginning of a project. Several definitions for CIM is given in this chapter, but Singh (1996) highlights all the relevant issues (**printed in bold**) in the following definition:

“CIM is the integration of the total manufacturing enterprise through the use of integrated systems and data communications coupled with new managerial philosophies that prove organisational and personnel efficiency.”

These CIM principles were found applicable to the integration of the subsets of MOBROB and highlighted some of the mistakes made in the past with regards to the ease of integration.

The second literature chapter is a survey on mobile robotics, concentrating on the issues in mobile robotics used in the MOBROB project.

Mobile robot design, positioning, mapping, path-planning, and communication with mobile robots were discussed. All the literature is for industrial type of mobile robots used indoor. Robots for outdoor use are not discussed at all. The author found the literature on this type of mobile robot very interesting, but not that applicable to the project.

The five subsets of the MOBROB project was studied and discussed in detail. This chapter is the main part of the literature survey.

This literature equipped the author with the necessary expertise to perform rework on the subsets where necessary and then to integrate them completely.

13.2 THE INTEGRATED CONTROL SYSTEM

The integrated control system proposed in this document is software based and difficult to separate from the existing subsets. Where applicable the subsets were changed to make them easier to integrate rather than trying to integrate them exactly as they were. In some cases, this would have been impossible, since some of the subsets used the same hardware.

The control system consists of three sub-parts. Each of these parts are developed in a different computer language, namely:

- TURBO PASCAL,
- BORLAND DELPHI, and
- MATLAB.

The author found that DOS based languages controls hardware better than Windows® based languages. It is however difficult to create a user friendly interface in these languages and it was therefore used for the onboard programs, where proper hardware control was important and a user friendly interface not that important. TURBO PASCAL was the language of choice here.

On the base station, two separate languages were used. BORLAND DELPHI was used to create the user interface and send files to the robot, while MATLAB were used for mathematical calculations and to create the files with the instructions for the robot.

One of the disadvantages of this system is that it is difficult to create an executable program from MATLAB code and MATLAB must be installed on the base station.

The control system developed was implemented on the MOBROB project successfully.

13.3 THE FUTURE OF MOBROB

There are quite a few subsets that still need to be developed and implemented. Docking and the execution of tasks are two examples.

The author would recommend that a new vehicle is build before any more subsets are developed. A circular shaped robot chassis will be much easier to control, dock and steer and will be able to make sharper turns, resulting in narrower aisles needed for the robot to drive in. The fuzzy rules may need some rethinking as well.

14 REFERENCES

- 1) Berenji, Hamid R., 1993, Fuzzy and Neural Control, An Introduction to intelligent and autonomous control, Kluwer Academic Publishers, The Netherlands.
- 2) Borenstein J., Everett H.R., Feng L., Wehe D., 1997, Mobile Robot Positioning: Sensors and Techniques, *Journal of Robotic Systems*, 14(4), pp. 231-249, John Wiley & Sons, Inc.
- 3) Borenstein J., Feng L., 1995, UMBmark: A Benchmark Test for Measuring Odometry Errors in Mobile Robots, *Presented at the 1995 SPIE Conference on Mobile Robots, Philadelphia*, October 22-26.
- 4) Borenstein J., Feng L., 1996, Gyrodometry: A new Method for Combining Data from Gyros and Odometry in Mobile Robots, *Proceedings of the 1996 IEEE International Conference on Robotics and Automation, Minneapolis*, Apr. 22-28, pp 423-428.
- 5) Borenstein J., Koren Y., 1991, The Vector Field Histogram – Fast Obstacle Avoidance for Mobile Robots, *IEEE Journal of Robotics and Automation* Vol 7, No 3, June 1991, pp 278-288.
- 6) Connessons, E. & Vasiljevic, C., 2000, Absolute location by landmark extraction. *Robotica* volume 18, pp 487 – 493. Cambridge University Press.
- 7) Deist L.A., 1993, Fuzzy Logic controller for an autonomous guided vehicle, M.Eng thesis, Industrial Engineering, University of Stellenbosch.
- 8) Hoppenot, P., Colle, E. & Barat, C., 2000, Off-line localisation of a mobile robot using ultrasonic measurements, *Robotica* volume 18, pp 315 – 323. Cambridge University Press.
- 9) Koenig, S., Tovey, C. & Halliburton, W., 2001, Greedy Mapping of Terrain, College of Computing, Georgia Institute of Technology, On-line publications. <http://www.cc.gatech.edu/ai/robot-lab/publications.html>.

- 10) Meng, Q., Sun Y. & Cao, Z., 2000, Adaptive Extended Kalman filter (AEKF)-based mobile robot localisation using sonar. *Robotica* volume 18, pp 459 – 473. Cambridge University Press.
- 11) Mentz, L., 1997, A Study of the Different Navigational Techniques for a Mobile Robot, B.Ing final year project report, Industrial Engineering, University of Stellenbosch.
- 12) Nel, A.J., 1997, The Design and Construction of a Mobile Robot for Material Handling, M.Ing thesis, Industrial Engineering, University of Stellenbosch.
- 13) Ramírez, G. & Zeghloul, S., 2001, Collision-free path-planning for nonholonomic mobile robots using a new obstacle representation in the velocity space. *Robotica* volume 19, pp 543 – 555. Cambridge University Press.
- 14) Ránky, P.G., 1986, *Computer Integrated Manufacturing*. Book. Prentice Hall, Inc.
- 15) Salichs, M.A. & Moreno, L., 2000, Navigation of Mobile Robots: Open Questions. *Robotica* volume 18, pp 227 – 234. Cambridge University Press.
- 16) Saffiatti, A., 1997. The uses of fuzzy logic in Autonomous Robot Navigation. *Soft Computing* 1(4): 180-197.
- 17) Singh, N. 1996. *Systems Approach to Computer-Integrated Design and Manufacturing*. Book. John Wiley & Sons, Inc.
- 18) Steenkamp, M.L., 1998, Communication with a Mobile Robot using Radio Frequencies, B.Ing final year project, Industrial Engineering, University of Stellenbosch.
- 19) Stremler, F.G., 1990, *Introduction to Communication Systems*, pp 127.
- 20) Vajpayee, S.K, 1995, *Principles of Computer Integrated Manufacturing*. Book. Prentice Hall, Inc.
- 21) Van Rooyen, M., 1997, The Mapping and Path-planning of an Industrial Mobile Robot, B.Ing final year project report, Industrial Engineering, University of Stellenbosch.

- 22) Waldner, J., 1990, CIM Principles of Computer Integrated Manufacturing. Book. John Wiley & Sons, Inc.
- 23) Wang, Y., Lane, D.M. & Falconer, G.J., 2000, Two Novel approaches for unmanned underwater vehicle path planning: constrained optimisation and semi-infinite constrained optimisation. Robotica volume 18, pp 123 – 143. Cambridge University Press.
- 24) Williams, M. & Jones, D.I., 2001, A rapid method for planning paths in three dimensions for a small aerial robot. Robotica volume 19, pp 125 – 135. Cambridge University Press.

15 BIBLIOGRAPHY

- 1) Berenji, Hamid R., 1993, Fuzzy and Neural Control, An Introduction to intelligent and autonomous control, Kluwer Academic Publishers, The Netherlands.
- 2) Borenstein J., Everett H.R., Feng L., Wehe D., 1997, Mobile Robot Positioning: Sensors and Techniques, *Journal of Robotic Systems*, 14(4), pp. 231-249, John Wiley & Sons, Inc.
- 3) Borenstein J., Feng L., 1995, UMBmark: A Benchmark Test for Measuring Odometry Errors in Mobile Robots, *Presented at the 1995 SPIE Conference on Mobile Robots, Philadelphia*, October 22-26.
- 4) Borenstein J., Feng L., 1996, Gyrodometry: A new Method for Combining Data from Gyros and Odometry in Mobile Robots, *Proceedings of the 1996 IEEE International Conference on Robotics and Automation, Minneapolis*, Apr. 22-28, pp 423-428.
- 5) Borenstein J., Koren Y., 1991, The Vector Field Histogram – Fast Obstacle Avoidance for Mobile Robots, *IEEE Journal of Robotics and Automation* Vol 7, No 3, June 1991, pp 278-288.
- 6) Borland Delphi 3 for Windows 95 & Windows NT, Developer's Guide.
- 7) Connessons, E. & Vasiljevic, C., 2000, Absolute location by landmark extraction. *Robotica* volume 18, pp 487 – 493. Cambridge University Press.
- 8) Deist L.A., 1993, Fuzzy Logic controller for an autonomous guided vehicle, M.Eng thesis, Industrial Engineering, University of Stellenbosch.
- 9) Fraser, C. & Milne, J., 1994, Integrated Electrical and Electronic Engineering for Mechanical Engineers. Book. McGraw-Hill International Limited.
- 10) Hoppenot, P., Colle, E. & Barat, C., 2000, Off-line localisation of a mobile robot using ultrasonic measurements, *Robotica* volume 18, pp 315 – 323. Cambridge University Press.
- 11) Horowitz, P. & Hill, W., 1989, The Art of Electronics, Second Edition. Book. Cambridge University Press.
- 12) Koenig, S., Tovey, C. & Halliburton, W., 2001, Greedy Mapping of Terrain, College of Computing,

Georgia Institute of Technology, On-line publications. <http://www.cc.gatech.edu/ai/robot-lab/publications.html>.

- 13) Meng, Q., Sun Y. & Cao, Z., 2000, Adaptive Extended Kalman filter (AEKF)-based mobile robot localisation using sonar. *Robotica* volume 18, pp 459 – 473. Cambridge University Press.
- 14) Mentz, L., 1997, A Study of the Different Navigational Techniques for a Mobile Robot, B.Ing final year project report, Industrial Engineering, University of Stellenbosch.
- 15) Nel, A.J., 1997, The Design and Construction of a Mobile Robot for Material Handling, M.Ing thesis, Industrial Engineering, University of Stellenbosch.
- 16) Ramírez, G. & Zeghloul, S., 2001, Collision-free path-planning for nonholonomic mobile robots using a new obstacle representation in the velocity space. *Robotica* volume 19, pp 543 – 555. Cambridge University Press.
- 17) Ráanky, P.G., 1986, *Computer Integrated Manufacturing*. Book. Prentice Hall, Inc.
- 18) Saffiatti, A., 1997. The uses of fuzzy logic in Autonomous Robot Navigation. *Soft Computing* 1(4): 180-197.
- 19) Salichs, M.A. & Moreno, L., 2000, Navigation of Mobile Robots: Open Questions. *Robotica* volume 18, pp 227 – 234. Cambridge University Press.
- 20) Singh, N. 1996. *Systems Approach to Computer-Integrated Design and Manufacturing*. Book. John Wiley & Sons, Inc.
- 21) Steenkamp, M.L., 1998, Communication with a Mobile Robot using Radio Frequencies, B.Ing final year project, Industrial Engineering, University of Stellenbosch.
- 22) Stremler, F.G., 1990, *Introduction to Communication Systems*, pp 127.
- 23) The Math Works Inc., 1995, *The Student Edition of MATLAB, The Ultimate Computing Environment for Technical Education, Ver 4 User's Guide*. Book. Prentice Hall Inc.
- 24) Vajpayee, S.K., 1995, *Principles of Computer Integrated Manufacturing*. Book. Prentice Hall, Inc.
- 25) Van Rooyen, M., 1997, The Mapping and Path-planning of an Industrial Mobile Robot, B.Ing final year

project report, Industrial Engineering, University of Stellenbosch.

- 26) Wakerly, J.F., 1994, Digital Design Principles and Practices, Second Edition. Book. Prentice Hall, Inc.
- 27) Waldner, J., 1990, CIM Principles of Computer Integrated Manufacturing. Book. John Wiley & Sons, Inc.
- 28) Wang, Y., Lane, D.M. & Falconer, G.J., 2000, Two Novel approaches for unmanned underwater vehicle path planning: constrained optimisation and semi-infinite constrained optimisation. Robotica volume 18, pp 123 – 143. Cambridge University Press.
- 29) Williams, M. & Jones, D.I., 2001, A rapid method for planning paths in three dimensions for a small aerial robot. Robotica volume 19, pp 125 – 135. Cambridge University Press.
- 30) Wyatt, A.L., Sr., 1992, Using Assembly Language, Third Edition. Book. Que® Corporation.

APPENDIX A: MOBROB SUBSET SYNOPSIS BY THE DIFFERENT AUTHORS

In this APPENDIX the synopsis of every subset of the MOBROB project, as it was written by the different authors is given. This should give the reader perspective on the chaotic state of the MOBROB project at the beginning of this project.

COMMUNICATION WITH A MOBILE ROBOT USING RADIO FREQUENCIES (STEENKAMP, 1998)

This project is about the mobile robot (MOBROB) and communicating with the robot via radio frequencies. It also includes the implementation of such a system.

Firstly an overview of MOBROB is given that will give the reader an idea of the work that was previously done on the robot. A review of the applications, mechanical design, navigation, positioning, sensors and communication system of MOBROB is given.

The best way of communicating with MOBROB was found to be through radio frequencies. The use of Radiometrix's BiM433-F Receive-Sender is investigated. The modules were bought and implemented with the necessary software on MOBROB.

THE MAPPING AND PATH-PLANNING OF AN INDUSTRIAL MOBILE ROBOT (VAN ROOYEN, 1997)

In the quest for more flexible manufacturing systems aiming to realise higher productivity, flexibility, as well as higher quality – the key factor is a system that can change quickly and easily in order to service a varying demand. The use of the industrial mobile robot supports this goal by providing the flexibility and integration required for flexible materials handling systems.

The first objective of the project was to develop a path-planning technique for a mobile robot. In order to fulfil this requirement, a suitable presentation of the factory layout was needed. This model of the layout

needed to be very flexible and easily updateable. The creation of this presentation of the layout became the second objective of the project.

At first, the model representing the factory floor was developed. A potential field map of a bounded two-dimensional region containing a goal location, and an arbitrary number of obstacles was used to represent the factory floor.

The potential field map description of the region can be used by an autonomous mobile robot to guide itself from any location to a goal location while avoiding any obstacles present. Path-planning is a matter of working out a route over which the potential value drops the fastest.

The development model of the factory layout ensured a very flexible system, which proved to be able to handle varying conditions on the factory floor.

The result of the project is that both main objectives were adequately met.

The conclusion of this report is that the potential field method developed proved to be very flexible and effective.

It is recommended that further effort be spent in developing a central server in order to determine the sequence in which service points should be visited. This central server should be a stationary computer using high frequency control to communicate with the robot.

A STUDY OF THE DIFFERENT NAVIGATIONAL TECHNIQUES FOR A MOBILE ROBOT (MENTZ, 1997)

In the building of MOBROB, the Industrial Engineering Department's mobile robot, a way had to be found to navigate it in order for it to be useful. The aim for this robot is to be able to move around in a known environment (in this case the laboratory) with known dimensions. It is imperative that the robot should know at any time, when required, where in this known environment it is. It should also know in which direction it is facing. The need for this information stems from the need to be able to navigate the robot from one point to another.

As detailed path-planning needs to be done in order to navigate the robot efficiently, it regularly needs an update of its position.

This paper addresses a few problems that were experienced while navigating the robot. The existing system consists of a mobile robot controlled with fuzzy logic and steered with dead-reckoning, which will be explained later in the paper. It also has sensors, which prevents it from bumping into objects along its way.

The further requirements were to determine the exact position in the environment (considering that the dead-reckoning is not always accurate) and the direction in which the front of the robot is facing. The system developed in this paper can thus be seen as more of a positioning system than a fully navigational system, as it treats only two aspects of the full navigational system. This navigational system can be broken down into three parts: Steering, Positioning and Path-planning.

All three parts put together will make the robot perform according to specifications.

In the first part of the paper, the different techniques for navigation will be discussed and a recommendation made on the best suitable method to use. In the second part of the paper, the validity of the solution will be tested. Again, the parts of this navigational system which have already been installed and are working, will not be addressed as it is the current problems that need to be solved.

THE DESIGN AND CONSTRUCTION OF A MOBILE ROBOT FOR MATERIAL HANDLING (NEL, 1997)

Mobile Robotics is a relative new field of research. Mobile robots differ from automated guided vehicles (AGV's) mainly because it is totally free-ranging. The main objectives of this project were to design and construct a mobile robot (MOBROB) from scratch. A fuzzy logic controller developed in a previous research project had to be implemented. A further objective was to control MOBROB from any point in a factory to any goal position, while side-stepping obstacles in its way. The final objective was to integrate a path-planning module and a navigational system with the MOBROB project.

In order to do a proper design of MOBROB, it was necessary to investigate all possible components which could be used on a mobile robot. The components which influenced the design the most were, the different types of steering arrangements, driving sources, power sources and control computers. Research has been done on positioning systems for mobile robots, and the basic concepts of fuzzy logic control. The fuzzy logic controller previously developed for the use on MOBROB was evaluated for its applicability.

MOBROB's component costs were the largest constraint during the design phase. Several suppliers were contacted to assure the best compromise between total cost and the overall performance of MOBROB. It was

found that the previously developed fuzzy logic controller could not be implemented directly unto MOBROB. It was therefore necessary to develop a new controller. After previous research on MOBROB had been considered, it was decided to represent its environment as a network of nodes and links. A path-planning module that can determine the shortest route between any two nodes was developed. A positioning system was developed to determine MOBROB's position (x, y, φ) by measurement of wheel rotation. During the final phase of the project, the navigation and path-planning modules and the fuzzy logic controller were integrated in a menu-driven user interface.

During evaluation of MOBROB, it was found that the representation of a factory floor as a network of nodes and links is efficient and flexible. MOBROB could be controlled from any node to any other node without collision with any of the known obstacles. Stopping prevented head-on collisions and dead-end routes could be identified. The performance of the path-planning unit was satisfactory, but it still required user intelligence, to prevent MOBROB from making U-turns.

Collisions between temporary obstacles and MOBROB's sides were the result of a lack of sensory input from the sides and rear. The user intelligence that is required during path-planning is the result of MOBROB's inability to make U-turns. The positioning system yielded significant errors, especially if long distances were travelled. These errors were caused by orientational errors owing to wheel slippage.

MOBROB's ability to avoid obstacles can be improved if the sensor system is expanded to give sensor readings for the sides and back of the vehicle. The positioning system can significantly be improved by the use of a gyroscope to measure the real-time vehicle orientation. The path-planning module can only function independently once MOBROB is allowed to make U-turns.

APPENDIX B: MATHEMATICAL CALCULATIONS

In this APPENDIX all the mathematical calculations and proofs are given.

MATHEMATICAL CALCULATIONS FOR THE GLOBAL NAVIGATIONAL SYSTEM

The mathematical proof for determining the robot's position is as follows:

A hypothetical situation is used to solve the problem. The location of the robot can be described mathematically in the following manner:

$$\frac{b}{\sin \beta} = \frac{r}{\sin \delta} \Rightarrow r = \frac{b \sin \delta}{\sin \beta}$$

$$\frac{a}{\sin \alpha} = \frac{r}{\sin \gamma} \Rightarrow r = \frac{a \sin \gamma}{\sin \alpha}$$

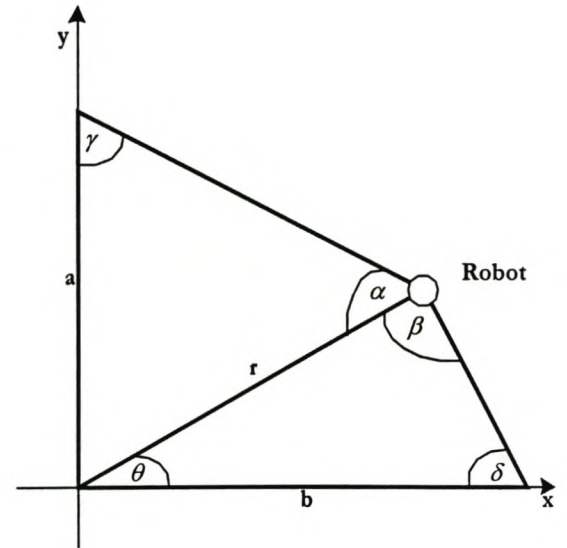
$$\therefore \frac{b \sin \delta}{\sin \beta} = \frac{a \sin \gamma}{\sin \alpha}$$

$$90^\circ + \gamma + \alpha + \beta + \delta = 360^\circ$$

$$\therefore \delta = 360^\circ - (90^\circ + \alpha + \beta + \gamma)$$

$$\sin \delta = -\sin(90^\circ + \alpha + \beta + \gamma)$$

$$\sin \delta = -\cos(\alpha + \beta + \gamma)$$



Now use the above results in combination to determine $\tan \gamma$

$$\begin{aligned} \therefore \frac{-b \cos(\alpha + \beta + \gamma)}{\sin \beta} &= \frac{a \sin \gamma}{\sin \alpha} \\ &= \frac{-b[\cos(\alpha + \beta) \cos \gamma - \sin(\alpha + \beta) \sin \gamma]}{\sin \beta} \end{aligned}$$

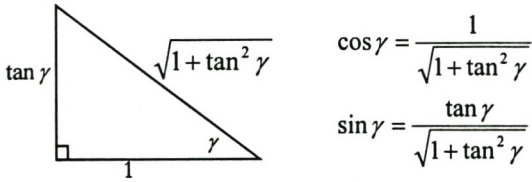
Take all the $\sin \gamma$ to one side

$$\therefore \sin \gamma \left[\frac{a}{\sin \alpha} - \frac{b}{\sin \beta} (\sin(\alpha + \beta)) \right] = \frac{-b}{\sin \beta} (\cos(\alpha + \beta) \cos \gamma)$$

$$\sin \gamma \left[\frac{a \sin \beta - b(\sin(\alpha + \beta)) \sin \alpha}{\sin \alpha \sin \beta} \right] = \left[\frac{-b(\cos(\alpha + \beta))}{\sin \beta} \right] \cos \gamma$$

$$\therefore \tan \gamma = \frac{b(\cos(\alpha + \beta)) \sin \alpha}{b(\sin(\alpha + \beta)) \sin \alpha - a \sin \beta}$$

Now write everything in terms of $\tan \gamma$



Now,

$$r = \frac{a \sin \gamma}{\sin \alpha}$$

$$\therefore r = \frac{a}{\sin \alpha} \left[\frac{\tan \gamma}{\sqrt{1 + \tan^2 \gamma}} \right]$$

Now use the internal angles for further manipulation

$$90^\circ + \theta = \alpha + \gamma$$

$$\sin(90^\circ + \theta) = \cos \theta$$

$$= \sin(\alpha + \gamma)$$

$$= \sin \alpha \cos \gamma + \cos \alpha \sin \gamma$$

$$\therefore \cos \theta = \sin \alpha \left[\frac{1}{\sqrt{1 + \tan^2 \gamma}} \right] + \cos \alpha \left[\frac{\tan \gamma}{\sqrt{1 + \tan^2 \gamma}} \right]$$

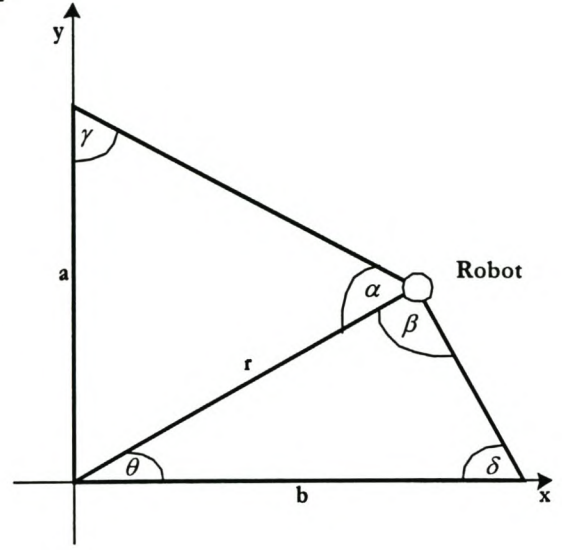
Also:

$$\cos(90^\circ + \theta) = -\sin \theta$$

$$= \cos(\alpha + \gamma)$$

$$= \cos \alpha \cos \gamma - \sin \alpha \sin \gamma$$

$$\therefore \sin \theta = \sin \alpha \left[\frac{\tan \gamma}{\sqrt{1 + \tan^2 \gamma}} \right] - \cos \alpha \left[\frac{1}{\sqrt{1 + \tan^2 \gamma}} \right]$$



Use all available information:

$$x = r \cos \theta$$

$$y = r \sin \theta$$

Now,

$$x = r \cos \theta$$

$$= \frac{a}{\sin \alpha} \left[\frac{\tan \gamma}{\sqrt{1 + \tan^2 \gamma}} \right] \frac{\sin \alpha + \cos \alpha \tan \gamma}{\sqrt{1 + \tan^2 \gamma}}$$

$$= \frac{a(\sin \alpha + \cos \alpha \tan \gamma) \tan \gamma}{\sin \alpha (1 + \tan^2 \gamma)}$$

$$= \frac{a(1 + \cot \alpha \tan \gamma) \tan \gamma}{1 + \tan^2 \gamma}$$

and

$$y = r \sin \theta$$

$$= \frac{a}{\sin \alpha} \left[\frac{\tan \gamma}{\sqrt{1 + \tan^2 \gamma}} \right] \frac{\sin \alpha \tan \gamma - \cos \alpha}{\sqrt{1 + \tan^2 \gamma}}$$

$$= \frac{a(\tan \gamma - \cot \alpha) \tan \gamma}{1 + \tan^2 \gamma}$$

This gives the final formulas:

$$x = \frac{a(1 + \cot \alpha \tan \gamma) \tan \gamma}{1 + \tan^2 \gamma}$$

$$y = \frac{a(\tan \gamma - \cot \alpha) \tan \gamma}{1 + \tan^2 \gamma}, \text{ where}$$

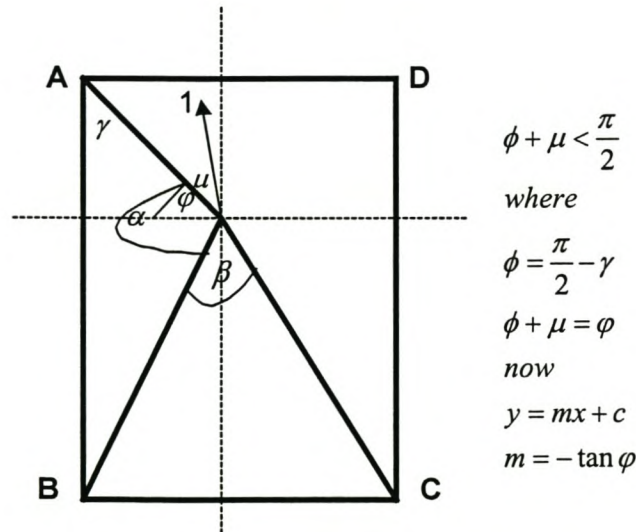
$$\tan \gamma = \frac{b(\cos(\alpha + \beta) \sin \alpha}{b(\sin(\alpha + \beta)) \sin \alpha - a \sin \beta}$$

In the above calculations, a , b , α , and β are known or can be easily calculated, therefore $\tan \gamma$ can always be calculated. This results in the values of x and y . These values were the objective of the above mathematical calculations.

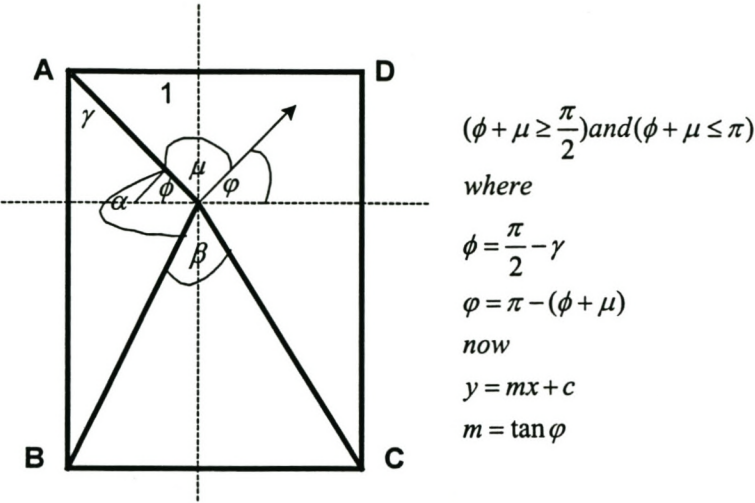
The next step in the positioning of the robot is to determine the direction the robot is facing. For the above example two different scenarios can exist when determining the direction the robot is facing. The first is when

$\phi + \mu < \frac{\pi}{2}$ and the second is when $(\phi + \mu \geq \frac{\pi}{2})$ and $(\phi + \mu \leq \pi)$.

The first scenario may be expressed as follows:



and the second scenario as follows:



This concludes the mathematical calculations for the global navigational subset of MOBROB.

APPENDIX C: MOBROB USER GUIDE

REQUIRED FILES

The control system requires quite a few files. A list of these files is given in this paragraph.

15.1.1 DELPHI FILES

Only one DELPHI file is needed. This is a program file and must be on the base station.

- MOBROB_Control.exe

15.1.2 MATLAB FILES

It was not possible to create an .exe file for the MATLAB programs and a whole list of files is required here.

First of all it is important that MATLAB is installed on the base station under the directory:

- C:\Matlab\, with the executable file MATLAB.exe in
- C:\Matlab\Bin\.

Other MATLAB files needed on the base station are:

- The MATLAB m files Main.m,
- Mainqp.m,
- Loadmap.m.
- Loadmapqp.m,
- Createmap.m,
- Editmap.m,
- Loadpos.m,

- Cream.m,
- Newpos.m,
- Posib.m,
- Potmap.m,
- Potmap2.m,
- Starget.m,
- Stargetqp.m,
- Walk.m, and
- Walkqp.m.
- The position file, Current.pos.
- The quick planning files wmap.mop,
- Wmap.pot,
- Wmap.pth, and
- Wmap.map.

15.1.3 PASCAL FILES

Two files are required on the onboard computer: one PASCAL-created .exe file and the position file,

- MOBCtrl.exe and
- Current.pos.

DIRECTORY STRUCTURE

It is important that the above listed files are in the correct directory structure for the programs to operate correctly.

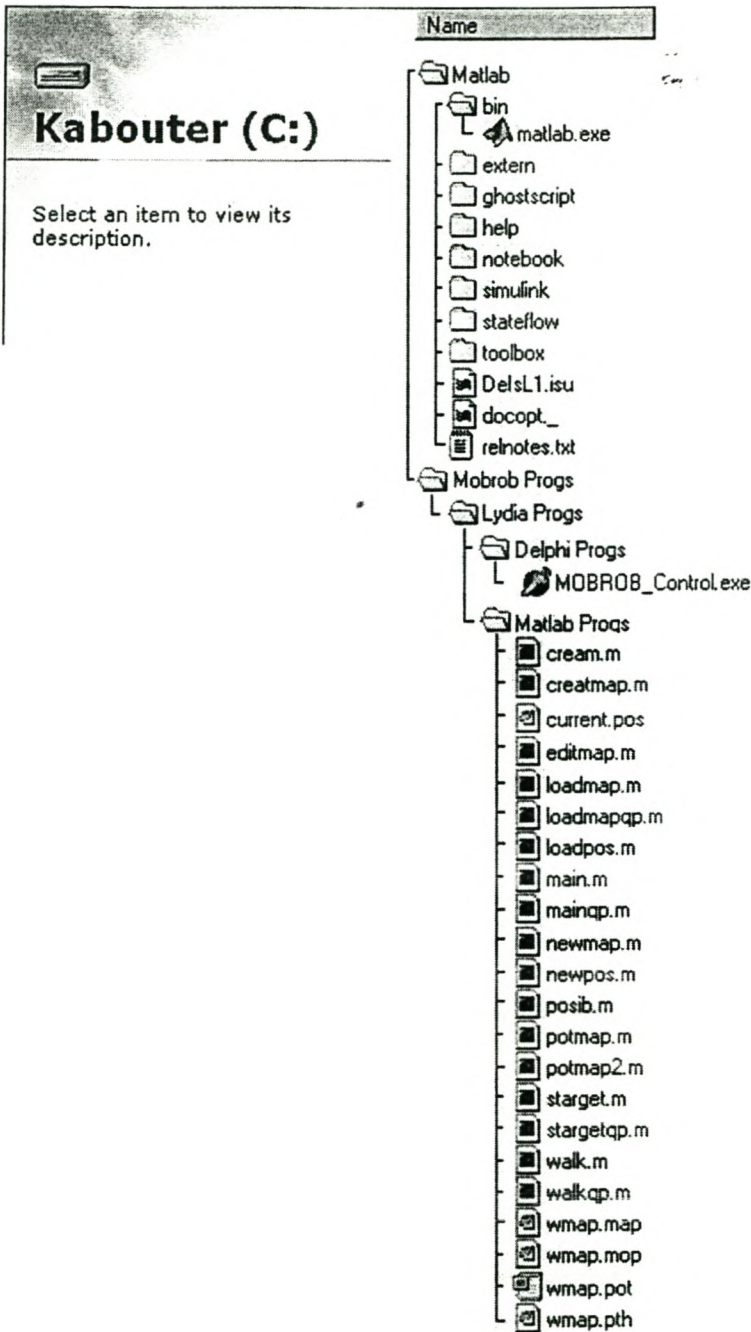
15.1.4 BASE STATION

The base station must have the directory structure shown in the figure on the next page.

15.1.5 ONBOARD COMPUTER

The onboard computer's files must be in the following directory.

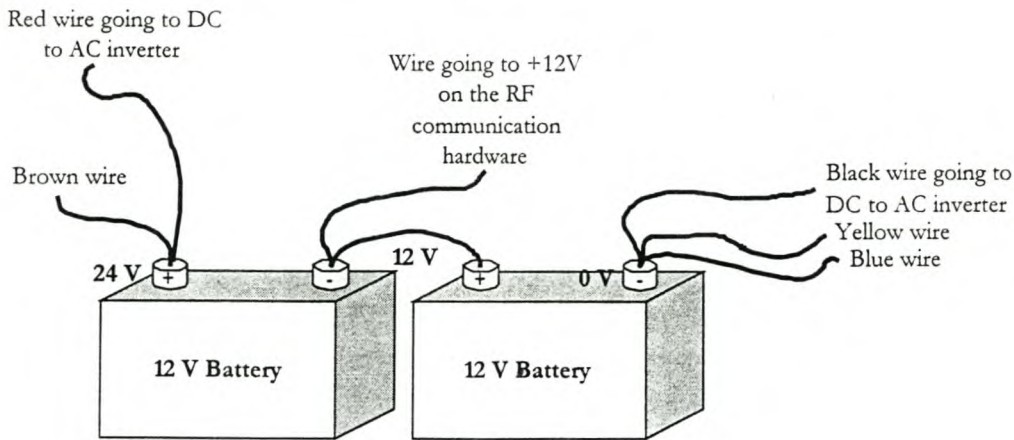
➡ C:\Mobrob\Lydia\.



DIRECTORY STRUCTURE REQUIRED FOR THE BASE STATION

MOBROB WIRING

All wires must be removed from the onboard batteries when the power is switched off. The following figure shows how to reconnect these wires.



WIRIN

G OF THE MOBROB ONBOARD POWER

It is important to remember that the two batteries must be connected to each other first, including the wire going to the RF communication hardware. When these wires are in place, all the wires connected to 0V must be connected. The 24V wires must be connected last and the user is warned that the pole of the battery may spark. When the wires are removed again, change the order and remove the red wire first. Make sure that the batteries are not short-circuited at any stage during this or any other process.

WALKTHROUGH

A thorough walkthrough is given in chapter eight of this document, therefore this paragraph will just be a brief description without illustrations.

- 1) A map must be created first. To create a map, click on the path-planning button in the Station Control program. In the path-planning program, click on the Create Map button, enter the factory floor dimensions and add obstacles to the map with the mouse. To remove an obstacle, click on it again with the mouse. When all the obstacles have been added, click on the red triangle with the mouse. After a while (this takes some time), a filename is asked. Select or enter a filename and save the file.
- 2) To edit a map, select path planning in the Station Control program and select edit map in the path-planning program. Select a map to edit, add and remove obstacles and click on the red triangle again, when finished. Again, the user will be prompted for a filename.

- 3) To plan a path in the path-planning program, click on the Load Map button, select a map, click the Set Target button, click on the target. If the large cross does not disappear, the destination was not valid and a new destination must be selected. A path is created and the user is prompted for a filename. The path is saved.
- 4) Quick path-planning can be done after path-planning has been done at least once on the map currently in use. The files are loaded and saved automatically.
- 5) To send a path to MOBROB with the Transmit button, select the *.map file of the path that must be sent to MOBROB. When prompted, switch the RF communication hardware on and to sending mode. This hardware must be switched back to receiving mode again when the user is reminded to do so.
- 6) The files that must be sent to MOBROB will be selected automatically when Quick Transmit is used.
- 7) On MOBROB, the MOBCtrl.exe program must be running, MOBROB must be wired correctly, the RF communication hardware as well as the sensors and driving motors must be switched on. MOBROB will drive to its destination. The user must enter the corners needed for the global positioning system when asked for it. When MOBROB reaches its destination, it will ask for the corners again and create the current.pos file. This file is then sent to the base station for future use.

APPENDIX D: MOBROB PASCAL PROGRAMS

The Pascal code created and edited are included in this appendix.

INTCTRL4.PAS

Intctrl stands for integrated control. This is the program code for the control program on the robot.

```
{E+,F+,N+,X+}
{$M 32000,0,655360}
PROGRAM Intctrl4;
USES
  CRT, GRAPH,           {Pascal system units}
  rtstdhdr, rtgsubs, rtgcommo, rtmouse, {Quinn-Curtis units}
  rttext, rtgwin, qcmenu, qcdialog,    {used for real time}
  rtgraph, rtbargr, rtmeter, rtcntrl,  {graphics}
  map, help1, sensor, fuzzy, control,  {user written units}
  dmcapi, dmcpub, dmcbus,              {DMC-620 control card units}
  rfu309;                             {RF Unit}
```

Const

```
DMC_base_address = 1000;
NodeLimit       = 12;
FactoryFloorMinY = 0;
FactoryFloorMinX = 0;
CurrentPosFile = 'c:\mobrob\Lydia\Current.pos';
MapFilePath = 'c:\mobrob\lydia\map.map';
NodePath = 'c:\mobrob\lydia\path.pth';
```

VAR

```
Simulation, MouseFound,
  PathViewOpen, Emergency, UserStop : Boolean;
TheMenu : MenuType;
ch : Char;
NodeNrString,
```

```

    dmcCode, ResponseString      : String;
dmc620                          : dmc_data;
EmergencyCounter, ResponseLen, code,
    FactMaxY, FactMaxX, MaxNodeNr,
    Result, Buttons, NthNode,
    ToNodeNr, Count, LastGoalNode : Integer;
SteerAngle, Xpos, Ypos,
    CurrentHeadAngle, Xstart, Ystart,
    StartCHA                : Real;
FromNode                    : Array[0..NodeLimit]
                          of Integer;

{*****}

Procedure EmergencyStop(Button:Integer; ButtonStates:IntPNTR);
var
    pathplanned : boolean;

Begin
    pathplanned := true;
    If Simulation = False then
        Begin
            dmcReset(dmc620);
            dmcCommand(dmc620, 'STXY', ResponseString, ResponseLen);
            If EmergencyStopHelp <> 1 then {when cancel is clicked}
                Begin
                    PathPlanned := False;
                    Emergency := True;
                    Exit;
                end else
                Begin
                    Emergency := False;
                    DmcCommand(dmc620, 'AC40000,40000',
                                ResponseString, ResponseLen);
                    DmcCommand(dmc620, 'SP80000,80000',
                                ResponseString, ResponseLen);

```



```

    end;
    end else UserStop := True;
END; {of Procedure EmergencyStop}

{*****}

Procedure SetupRTGraph;

Var
    rt, MeterXpos, MeterYpos,
    MeterRadius, LOWalarm, HIalarm,
    SetPnt, pinum, rr, MinY, MaxY,
    CenterP, rnum          : RealType;
    NrTraces, grid, xdecs, ydecs,
    startarc, stoparc, decs,
    NrNeedles, NthTic, Nrsegments,
    Button, Rows, Cols      : Integer;
    LineColor, LineFill,
    metercolors, meterls    : rtIntArrayType;
    ratchf, MouseFound      : Boolean;
    XYTags, SensorTags, AngleTags : TagArrayType;
    title                   : TitleType;
    Xunits, Yunits, units   : TagType;
    alarmstrings, SensAlarm : Alarmstrarraytype;
    startvalues, butcols, textcols : Array[0..1]
                                of Integer;
    swtags                  : Array[0..1]
                                of _string40;
    filename                : string;

Begin
    {*** Resize the windows for Display ***}
    rtSetPercentWindow(rtStat[0],
        0, 0, 0.5, 0.5); {Position window}
    rtSetPercentWindow(rtStat[1],
        0.505, 0, 1, 0.5); {Orientation }
    rtSetPercentWindow(rtStat[2],

```

```

0, 0.51, 1, 0.75);{Sensors}
rtSetPercentWindow(rtStat[3],
0.51, 0.76, 1, 1);{Emergency button}
rtSetPercentWindow(rtStat[4],
0, 0.76, 0.5, 1);{Dist to Node graph}
rtSetWindowOptions(rtStat[4], rt_Border);
rtInitWindowColors(rtStat[4], Green, Green,
Red, Green, Green, WHITE, WHITE);
AlarmStrings[0] := "";
AlarmStrings[1] := "";
AlarmStrings[2] := "";
LOWalarm := 0; HIalarm := 0; SetPnt := 0;
{*****Display Current and Goal Position*****}
{Graph Display Info }
rt := 0.99; NrTraces := 2; grid := 0;
ratchf := False; xdecs := 0; ydecs := 0;
{Create Titles, Units, and Tag Labels }
XYTags[0] := 'Current Position';
XYTags[1] := 'Goal Position';
YUnits := 'X position [cm]';
XUnits := 'Y position [cm]';
Title := 'POSITION';
LineColor[0] := 15; LineFill[0] := EmptyFill;
LineColor[1] := 10; LineFill[1] := EmptyFill;
rtSetMouse(0,600); {move mouse away from graph display area}
rtInitWindowColors(rtStat[0], DarkGray, Red, LightBlue,
LightRed, Yellow, White, White);
{Put a border around the window }
rtSetWindowOptions(rtStat[0], RT_BORDER);
{Draw XY graph}
filename := mapfilepath;
GetMapInfo(filename, FactMaxY, FactMaxX, MaxNodeNr);
rtSetupXYgraph(rtStat[0], FactoryFloorminX, FactmaxX,
FactoryFloorminY, FactmaxY, 10000, rt,
NrTraces, grid, LOWalarm, HIalarm, SetPnt,
xdecs, ydecs, Title, Xunits, Yunits,

```

```

        XYTags, LineColor, LineFill, ratchf);
{*****Display Current Head- and Goal Steer Angle*****}
MeterColors[0] := LightGreen;
MeterLs[0] := 3;    {LONG NEEDLE}
MeterColors[1] := Yellow;
MeterLs[1] := 2;
AngleTags[0] := 'Current'; AngleTags[1] := 'Goal';
MeterXpos := 0.5;    MeterYpos := 0.5;
MeterRadius := 0.35;    NrSegments := 24;
Nthtic := 1;    NrNeedles := 2;
rtSetMeterOptions(rtStat[1], rt_MeterOutsideLabel);
rtInitWindowColors(rtStat[1], Blue, Blue, Red,
    Red, White, White, White);
rtSetWindowOptions(rtStat[1], rt_Border);
rtSetupMeter(rtStat[1], -360, 360, MeterXpos,
    MeterYpos, MeterRadius, 280,260,
    0, 0, 0, 0, 'Orientation', 'Degrees',
    AngleTags, AlarmStrings, Nrneedles,
    NrSegments, nthtic, metercolors, meterls);
{*****Display Left, Right, and Front Sensor Readings*****}
LineFill[0] := 1; LineColor[0] := Red;
LineFill[1] := 1; LineColor[1] := Green;
LineFill[2] := 1; LineColor[2] := LightBlue;
SensorTags[0] := 'Right Distance';
SensorTags[1] := 'Front Distance';
SensorTags[2] := 'Left Distance';
SensAlarm[0] := 'Object to Close';
NthTic := 3; miny := 0;    maxy := 300;
CenterP := 0; LowAlarm := 60;    HIAlarm := 310;
SetPnt := 800; Decs := 0;
rtInitWindowColors(rtStat[2], DARKGRAY, DARKGRAY,
    Red, DarkGray, DarkGray, WHITE, WHITE);
rtSetWindowOptions(rtStat[2], RT_BORDER);
rtSetupHBarGraph(rtStat[2], miny, maxy, centerp, LowAlarm,
    HIAlarm, setpnt, decs,
    'Sensor Readings', 'cm', SensorTags,

```



```

        SensAlarm, NthTic,
            LineColor, LineFill);
{*****Display Emergency Button *****}
ButCols[0] := LightRed;  TextCols[0] := WHITE;
ButCols[1] := Red;      TextCols[1] := WHITE;
swTags[0] := 'Emergency Stop';
swTags[1] := 'Emergency Stop';
rtSetWindowOptions(rtStat[3], rt_3Dinner);
Rows := 1; Cols := 1;
rtOpenButtonBank(rtStat[3], @swtags, NIL, NIL, @TextCols,
    @ButCols, Rows, Cols, @StartValues,
    Button_OnOff, 0.1,
    0.1, ", EmergencyStop);
rtDrawButtonBank(rtStat[3]);
End;{of Procedure SetupRTGraph}

Procedure SetUpDistToNextNodeGraph(MaxDistToNextNode:Real);
Var
    LineColor, LineFill      : rtIntArrayType;
    DistTags                  : TagArrayType;
    DistAlarmstrings          : Alarmstrarraytype;

Begin
    LineFill[0] := 1;      LineColor[0] := 1;
    DistTags[0] := ' ';
    DistAlarmStrings[0] := 'Very Near';
    rtSetupHBarGraph(rtStat[4], 0, MaxDistToNextNode, 0, 30,
        5, 5, 0,'Distance To Next Node', 'cm',
        DistTags, DistAlarmStrings, 1,
        LineColor, LineFill);
end;{of Procedure SetUpDistToNextNodeGraph}

{*****}

Procedure UpdateDisplay(Xpos, Ypos, FrontDist, RightDist,
    LeftDist, GoalSteerAngle : real;
    EmergencyCounter : Integer;

```

```

        DistToGoal: Real);

Const
    DrawOffset = 5;

Var
    Pos, HeadAngles, SensorValues,
    GoalDistValues          : rtValueArrayType;
    GoalxPos, GoalyPos      : Real;

Begin
    {*****Update Current and Goal Position*****}
    Pos[0] := Xpos;  Pos[1] := Ypos;
    GetNodeCoords(FromNode[LastGoalNode], GoalxPos, GoalyPos);
    Pos[2] := GoalxPos - DrawOffset;
    Pos[3] := GoalyPos - DrawOffset;
    rtUpdateXYGraph(rtStat[0], Pos);
    Pos[2] := GoalxPos + DrawOffset;
    rtUpdateXYGraph(rtStat[0], Pos);
    Pos[3] := GoalyPos + DrawOffset;
    rtUpdateXYGraph(rtStat[0], Pos);
    Pos[2] := GoalxPos - DrawOffset;
    rtUpdateXYGraph(rtStat[0], Pos);
    Pos[3] := GoalyPos - DrawOffset;
    {*****Update Current Head- and Goal Steer Angle*****}
    HeadAngles[0] := CurrentHeadAngle;
    HeadAngles[1] := GoalSteerAngle;
    rtUpdateDisplay(rtStat[1], HeadAngles);
    {*****Update Left and Right Sensor Readings*****}
    SensorValues[0] := RightDist;
    SensorValues[1] := FrontDist;
    SensorValues[2] := LeftDist;
    rtUpdateDisplay(rtStat[2], SensorValues);
    {*****Check if Emergency Button pressed*****}
    rtprocesswindowoptions;
    {*****Update Near membership values *****}
    GoalDistValues[0] := DistToGoal;

```

```

    rtUpdateDisplay(rtStat[4], GoalDistValues);
End;{of Procedure UpdateDisplay}

```

```

{*****}

```

```

Procedure CloseRTGraph;

```

```

Begin

```

```

    rtCloseWindow(rtStat[0]);{Position window}
    rtCloseWindow(rtStat[1]);{Orientation }
    rtCloseWindow(rtStat[2]);{Sensors}
    rtCloseWindow(rtStat[3]);{Emergency button}
    rtCloseWindow(rtStat[4]);{NearMem graph}
    qcDrawMenu(TheMenu);

```

```

end;{of Procedure CloseRTGraph;}

```

```

{*****}

```

```

Procedure GetGoalSteerAngle(var GoalSteerAngle : real;

```

```

    Xpos, Ypos, NextNodeXPos, NextNodeYPos: Real);

```

```

Const

```

```

    SteerLimit = 0.001;

```

```

Var

```

```

    Deltax, Deltay, GoalHeadAngle      : Real;

```

```

Begin

```

```

    Deltax := NextNodeXPos-Xpos;

```

```

    Deltay := NextNodeYPos-Ypos;

```

```

    If (Deltay < SteerLimit) and (Deltay > -SteerLimit) then

```

```

    Begin

```

```

        if deltax > 0 then

```

```

            GoalHeadAngle := 90;  {To prohibited division by zero}

```

```

        If deltax < 0 then

```

```

            GoalHeadAngle := -90;

```

```

        end else          {First and Second Quadrant}

```

```

            GoalHeadAngle := 180/Pi*ArcTan((Deltax)/(Deltay));

```

```

        If deltay < 0 then    {3'rd quadrant} {4'th quadrant}

```



```

    GoalHeadAngle:=180+GoalHeadAngle;
GoalSteerAngle := GoalHeadAngle - CurrentHeadAngle;
If GoalSteerAngle > 360 then    {*****}
    GoalSteerAngle := GoalSteerAngle - 360; {To Ensure }
If GoalSteerAngle <-360 then    {-360<GoalSteerAngle<360}
    GoalSteerAngle := GoalSteerAngle + 360; {*****}
If GoalSteerAngle > 180 Then    {To ensure Mobrob takes }
    GoalSteerAngle := -360+GoalSteerAngle; {smallest goal }
If GoalSteerAngle <-180 Then    {steer angle for example -30}
    GoalSteerAngle := 360+GoalSteerAngle; {instead of +330}
End;{of Procedure GetGoalSteerAngle}

```

```

{*****}

```

```

Procedure AdjustHeadAngle;
Begin
    If CurrentHeadAngle >= 360 then
        CurrentHeadAngle := CurrentHeadAngle - 360;
    If CurrentHeadAngle <=-360 then
        CurrentHeadAngle := CurrentHeadAngle + 360;
    CurrentHeadAngle := CurrentHeadAngle + Steer.Angle;
end;{of Procedure AdjustHeadAngle}

```

```

{*****}

```

```

Procedure GetNodePath(NodePath : String);

```

```

Var

```

```

Continue      : Char;
PathFile      : Text;
Code, NodeNr  : Integer;

```

```

Begin

```

```

    Assign(PathFile, NodePath);
    Reset(PathFile);
    NodeNr := -1;
    Repeat
        NodeNr := NodeNr + 1;

```

```

Readln(PathFile,NodeNrString);
val(NodeNrString,FromNode[NodeNr], code);
Until NodeNrString = 'END';
Close(PathFile);
LastGoalNode := NodeNr-1;
End;{of Procedure GetNodePath;}

{*****}

Procedure SimulateDriveToNextNode;
Var
  FrontDist, LeftDist, RightDist,
  GoalSteerAngle, NextNodeXPos, NextNodeYPos,
  DistToGoal, MoveDist          : Real;
  NodeReached                   : Boolean;

Begin
  DistToGoal := 0;
  NodeReached := False;
  GetNodeCoords(ToNodeNr, NextNodeXPos, NextNodeYPos);
  Repeat
    Randomize; {generate random values for sensor readings}
    RightDist := 100 + Random(100);
    LeftDist := 100 + Random(100);
    FrontDist := 100 + Random(100);
    GetGoalSteerAngle(GoalSteerAngle, Xpos, Ypos,
                     NextNodeXPos, NextNodeYPos);
    CalcMembership(FrontDist, RightDist, LeftDist,
                  GoalSteerAngle, DistToGoal, SteerAngle, MoveDist,
                  Simulation);
    rtProcessWindowOptions;
    CalcNewPosition(SteerAngle, MoveDist, CurrentHeadAngle, Xpos, Ypos);
    AdjustHeadAngle;
    CheckNearNextNode(Xpos, Ypos, NextNodeXPos, NextNodeYPos,
                     DistToGoal,NodeReached);

    UpdateDisplay(Xpos, Ypos, FrontDist, RightDist,

```

```

    LeftDist, GoalSteerAngle, EmergencyCounter,
        DistToGoal);
rtProcessWindowOptions;
Delay(300);
Until (NodeReached = True) or (UserStop = True);
End;{of Procedure SimulateDriveToNextNode}

{*****}
Function Simulator:integer;

Var
    MaxDistToNextNode,
    GoalSteerAngle, NextNodeXpos, NextNodeYpos : Real;
    NodeReached, pathplanned : Boolean;

Begin
    pathplanned := true;
    If MapFilePath = " then
    Begin
        LoadFileFirstHelp;
        Exit;
    end;
    If PathPlanned = False then
    Begin
        PlanPathFirstHelp;
        Exit;
    end;
    rtCloseWindow(rtStat[5]);
    PathViewOpen := False;
    NthNode := 0;
    GetNodePath(NodePath);
    ToNodeNr := FromNode[NthNode];
    GetNodeCoords(ToNodeNr,NextNodeXPos, NextNodeYPos);
    GetGoalSteerAngle(GoalSteerAngle, Xpos, Ypos,
        NextNodeXPos, NextNodeYPos);
    CheckNearNextNode(Xpos, Ypos, NextNodeXPos, NextNodeYPos,

```



```

        MaxDistToNextNode,NodeReached);
If NodeReached = False then NthNode := NthNode-1;
Simulation := True;
SetupRTGraph;
Repeat
    ToNodeNr := FromNode[NthNode+1];
    GetNodeCoords(ToNodeNr,NextNodeXPos, NextNodeYPos);
    MaxDistToNextNode := sqrt(sqr(NextNodeXpos-Xpos) +
        sqr(NextNodeYpos-Ypos));
    SetUpDistToNextNodeGraph(MaxDistToNextNode);
    SimulateDriveToNextNode;
    NthNode := NthNode + 1;
Until (NthNode = LastGoalNode) or (UserStop = True);
{ SimulationEndHelp;}
UserStop := False;
CloseRTGraph;
Emergency := False;
Xpos := Xstart;
Ypos := Ystart;
CurrentHeadAngle := StartCHA;
Simulation := False;
end;{of Function Simulator}

{*****}
Procedure WaitForMoveEnd;
Begin
    Repeat
        rtProcessWindowOptions;
        Delay(50);
        dmc_bus_submit_legal_command(dmc620, 'AMX');
        dmc_bus_submit_legal_command(dmc620, 'AMY');
    until Dmc620.Response_string = '!';
end;{of Procedure WaitForMoveEnd}
{*****}
Procedure DriveToNextNode;

```

```

Var
  FrontDist, LeftDist, RightDist,
  GoalSteerAngle, NextNodeXPos, NextNodeYPos,
  DistToGoal, MoveDist          : Real;
  NodeReached                    : Boolean;
  EmergencyCount                 : Integer;
  InputChar                      : Char;
  Command                        : String;

Begin
  SteerAngle := 0;
  NodeReached := False;
  GetNodeCoords(ToNodeNr, NextNodeXPos, NextNodeYPos);
  EmergencyCount := 0;
  Repeat
    rtProcessWindowOptions;
    GetSensorReadings(FrontDist, RightDist, LeftDist);
  rtProcessWindowOptions;
    GetGoalSteerAngle(GoalSteerAngle, Xpos, Ypos,
                      NextNodeXPos, NextNodeYPos);
  rtProcessWindowOptions;
    StopFuzzyRules(Emergency);
    CalcMembership(FrontDist, RightDist, LeftDist,
                  GoalSteerAngle, DistToGoal, SteerAngle, MoveDist,
                  Simulation);
  rtProcessWindowOptions;
  If (Emergency = True) then
    Begin
      Emergency := False;
      EmergencyCount := EmergencyCount + 1;
      If EmergencyCount > 0 then
        Begin
          If EmergencyStopHelp <> 1 then {when cancel is clicked}
            Begin
              Emergency := True;
            Exit;

```

```

    end;
  end;
end else EmergencyCount := 0;
If Emergency = False then
Begin
  GetMoveCommand(SteerAngle, MoveDist, Command);
rtProcessWindowOptions;
  DmcCommand(dmc620, Command, ResponseString, ResponseLen);
  DmcCommand(dmc620, 'BGXY', ResponseString, ResponseLen);
rtProcessWindowOptions;
rtProcessWindowOptions;
  CalcNewPosition(SteerAngle, MoveDist, CurrentHeadAngle, Xpos, Ypos);
rtProcessWindowOptions;
  AdjustHeadAngle;
rtProcessWindowOptions;
  CheckNearNextNode(Xpos, Ypos, NextNodeXPos,
    NextNodeYPos, DistToGoal, NodeReached);
rtProcessWindowOptions;
  UpdateDisplay(Xpos, Ypos, FrontDist, RightDist,
    LeftDist, GoalSteerAngle, EmergencyCounter,
      DistToGoal);
  WaitForMoveEnd;
end;
Until (NodeReached = True) or (Emergency = True);
End; {of Procedure DriveToNextNode}

```

```

{*****}

```

```

Function ControlMOBROB:integer;

```

```

Var

```

```

  MaxDistToNextNode, GoalSteerAngle, NextNodeXpos, NextNodeYpos : Real;

```

```

  NodeReached, pathplanned : Boolean;

```

```

Begin

```

```

  pathplanned := true;

```

```

  If MapFilePath = " then

```

```

  Begin

```



```

LoadFileFirstHelp;
Exit;
end;
If PathPlanned = False then
Begin
  PlanPathFirstHelp;
  Exit;
end;
rtCloseWindow(rtStat[5]);
PathViewOpen := False;
If StartMOBROBHelp <> 1 then exit; {when cancel is clicked}
InitSensorHardware;
DmcInit600(dmc620, DMC_base_address);
DmcReset(dmc620);
DmcCommand(dmc620, 'AC30000,30000',
            ResponseString, ResponseLen);
DmcCommand(dmc620, 'SP30000,30000',
            ResponseString, ResponseLen);
NthNode := 0;
GetNodePath(NodePath);
ToNodeNr := FromNode[NthNode];
GetNodeCoords(ToNodeNr, NextNodeXPos, NextNodeYPos);
GetGoalSteer.Angle(GoalSteer.Angle, Xpos, Ypos,
                   NextNodeXPos, NextNodeYPos);
CheckNearNextNode(Xpos, Ypos, NextNodeXPos, NextNodeYPos,
                  MaxDistToNextNode, NodeReached);
If NodeReached = False then NthNode := NthNode+1;
SetupRTGraph;
Repeat
  ToNodeNr := FromNode[NthNode+1];
  GetNodeCoords(ToNodeNr, NextNodeXPos, NextNodeYPos);
  MaxDistToNextNode := sqrt(sqr(NextNodeXpos-Xpos) +
                           sqr(NextNodeYpos-Ypos));
  SetupDistToNextNodeGraph(MaxDistToNextNode);
  DriveToNextNode;
  If (NthNode < LastGoalNode-1) and (Emergency = False) then

```

```

    GetCurrentPos(Xpos, Ypos, CurrentHeadAngle,Factmaxx,FactmaxY);
    NthNode := NthNode + 1;
    Until (NthNode = LastGoalNode) or (Emergency = True);
    dmcCommand(dmc620, 'STXY', ResponseString, ResponseLen);
    dmcClose(dmc620);
    If Emergency = False then LastNodeReachedHelp;
    CloseRTGraph;
    PathPlanned := False;
    Emergency := False;
end;{of Function ControlMOBROB}

```

```

{*****}

```

```

Function QuitYesProc : Integer;

```

```

Begin

```

```

    quityesproc := 2;

```

```

End;{of Function QuitYesProc}

```

```

Procedure LoadCurrentPos;

```

```

var

```

```

    Currentpos : Text;

```

```

    Inputstr : String;

```

```

    strlength : integer;

```

```

    i,counter : integer;

```

```

    num : integer;

```

```

    newstr,dumstr : string;

```

```

    startystr, startxstr, dirstr : string;

```

```

    startdir : integer;

```

```

    code : integer;

```

```

    filename : string;

```

```

Begin

```

```

Assign(Currentpos,CurrentPosFile);
Reset(currentpos);
readln(currentpos,Inputstr);
    { The current.pos file has only one line }
strlength := length(Inputstr);
Close(currentpos);      { Close the file }
counter := 1;           { Initialise counter }
num := 1;               { Initialise num }
newstr := "";           { Initialise newstr }
for i := 1 to strlength do begin
    if Inputstr[i] <> ' ' then begin
        dumstr := Inputstr[i];
        newstr := newstr + dumstr;
        dumstr := "";
    end
else begin
    counter := 1;
    case num of
        1 : startystr := newstr;
        2 : startxstr := newstr;

    end;
    inc(num);
    newstr := "";
end;
end;
dirstr := newstr;

val(startystr,ystart,code);
val(startxstr,xstart,code);
val(dirstr,startdir,code);
case startdir of
    1 : currentheadangle := -90;
    2 : currentheadangle := -45;
    3 : currentheadangle := 0;
    4 : currentheadangle := 45;

```



```

5 : currentheadangle := 90;
6 : currentheadangle := 135;
7 : currentheadangle := 180;
8 : currentheadangle := -135;

end;

XPos := Xstart;
YPos := Ystart;
startCHA := currentheadangle;
filename := mapfilepath;
GetMapInfo(filename,FactMaxY,FactMaxX,MaxNodeNr);
end;

{*****}

Procedure EditCurrentPos;

var
  Currentpos : Text;
  xstr, ystr, dirstr : String;
  dir : integer;
  code : integer;
  filename : string;
  x, y : integer;
  modules : integer;

Begin
  x := round(xpos);
  y := round(ypos);
  modules := x mod 20;
  if modules < 10 then x := x - modules
  else x := x + (20 - modules);

  modules := y mod 20;
  if modules < 10 then y := y - modules
  else y := y + (20 - modules);

```

```

str(x,xstr);
str(y,ystr);
currentheadangle:=round(currentheadangle);
if currentheadangle < 0 then currentheadangle := 360 + currentheadangle;
if ((currentheadangle<293) and (currentheadangle>247)) then dir := 1;
if ((currentheadangle>292) and (currentheadangle<338)) then dir := 2;
if ((currentheadangle>337) or (currentheadangle<23)) then dir := 3;
if ((currentheadangle>22) and (currentheadangle<68)) then dir := 4;
if ((currentheadangle<113) and (currentheadangle>67)) then dir := 5;
if ((currentheadangle<158) and (currentheadangle>112)) then dir := 6;
if ((currentheadangle<203) and (currentheadangle>157)) then dir := 7;
if ((currentheadangle<248) and (currentheadangle>202)) then dir := 8;
str(dir,1,dirstr);
Assign(Currentpos,CurrentPosFile);
Rewrite(currentpos);
write(currentpos,ystr,' ',xstr,' ',dir);

      { The current.pos file has only one line }
Close(currentpos);      { Close the file }
end;

```

```

function Maincontrol:integer;
var
  mapname : string;
  pathname : string;
  currentpos : string;

```

```

BEGIN
  mapname := mapfilepath;
  pathname := nodepath;
  currentpos := currentposfile;
  Ontvangler(currentpos);
  Ontvangler(mapname);
  Ontvangler(pathname);

```

```

LoadCurrentPos;
Simulator;

ControlMOBROB;
getcurrentpos(xpos,ypos,currentheadangle,FactMaxX,FactMaxY);
EditCurrentPos;
Stuurler(currentpos);

end;

{*****}
Procedure InitTheMenu(Var am: MenuType);

Var
  Colors : RTIntArrayType;

Begin
  colors[0] := Blue;   colors[1] := white;
  colors[2] := white;  colors[3] := red;
  qcOpenMenu(am,1,1,60, colors);
  qcAddMenuItem(am,'FILE   ', Kbd_Null,0,NILMenuFunc,"");
  qcAddMenuItem(am,'Start ALT-L', Kbd_ALT_L,0,MainControl,"");
  qcAddMenuItem(am,'Quit ALT-Q', Kbd_ALT_Q,0,QuitYesProc,"");
END;{of Procedure InitTheMenu}

{*****}
Procedure OpenWindows;

Begin
  rtOpenrtG(DETECT, 0, defaultgraphdir, 8, 1);
  rtInitMouse(mousefound, buttons);

```



```

rtShowMouse;
InitTheMenu(TheMenu);
qcDrawMenu(TheMenu);
Result := 0;
Setbkcolor(blue);
end;{of Procedure OpenWindows}

{*****}

Procedure CloseWindows;

Begin
  qcCloseMenu(themenu);
  rtCloseGraphics(1);
end;{Procedure CloseWindows}

{*****}
{ * MAIN PROGRAM * }
{*****}

BEGIN

  OpenWindows;

  Repeat      {check for mouse events}
    If (qcCheckForMenuEvent(TheMenu) <> 0) then
      Result := qcProcessMenu(TheMenu)
    else
      begin
        If (KeyPressed) Then
          ch := ReadKey; {read input from keyboard }
        end;
        delay(50);
        Until (Result = 2); { if ans = 2 then Quit was selected }
      CloseWindows;
  END.

```

UNIT MAP

```
{ $m 32000,0,655360 }
{ $N+,E+,F+,X+ }
{ ***** }
Unit Map;
{ ***** }

{ ***** }
Interface
{ ***** }

Type
FileString = String[8];

Procedure Init.MapWindow(FactMaxY , FactMaxX,
                        MaxNodeNr: Integer; Title : String);
Procedure DelereSpaces(var InputString: FileString);
Procedure GetMapInfo(var MapFilePath: string; Var FactMaxY,
                    FactMaxX, MaxNodeNr: Integer);
Procedure DisplayNodes(MaxNodeNr: Integer);
Procedure DisplayLinks(MaxNodeNr: Integer);
Procedure SaveStartPosition(int: integer;
                          var instring: string);
Procedure InputStartPosition(var X,Y, CHA: Real; FactMaxY,
                          FactMaxX, MaxNodeNr: Integer;
                          var Error: Boolean);
Procedure GetCurrentPos(var X,Y, CHA: Real;MaxX,Maxy:real);
Procedure DoPathPlanning(MapFilePath: String; Var
                        NodePath: String);
Procedure GetNodeCoords(NthNode:integer; Var Xpos, Ypos:real);

{ ***** }
Implementation
{ ***** }
Uses
```

```

Help1, qcdialog, graph, rtstdhdr, rtgsubs,
    rtgwin, rtgraph, Fuzzy,
    navdrive;

```

```

Const

```

```

FactoryFloorMinY = 0;
FactoryFloorMinX = 0;
NodeLimit = 20; {must always be <30, because Quinn-Curtis can}
LinkLimit = 30; {only handle 32 traces per window}

```

```

Var

```

```

NrLinks, NodeNr          : array[0..NodeLimit]
                          of Integer;
x, y                     : array[0..NodeLimit]
                          of real;
LinkNodeNr               : array[0..NodeLimit, 1..LinkLimit, boolean]
                          of Integer;
Pos                       : rtvaluearraytype;
xpos, ypos, CurrentHeadAngle : Real;
StartNode, GoalNodeNr    : Integer;
Alpha, Beta, mu          : Real;
Scenario                  : Integer;

```

```

{*****}

```

```

Procedure GetMapInfo(var MapFilePath :string;
    Var FactMaxY , FactMaxX, MaxNodeNr : Integer);

```

```

Var

```

```

MapFile          : Text;
InputString      : FileString;

```



```

Code, RowNr, LinkCount      : Integer;
MapFileName                 : string;

Begin
  RowNr := -1 ;
  Assign(MapFile,MapFilePath);
  Reset(MapFile); Read(MapFile,InputString);
  Readln(MapFile,MapFileName); {Get the filename of the map}
  Read(MapFile,InputString);
  Readln(MapFile,InputString);
  DeleteSpaces(InputString);
  Val(InputString, MaxNodeNr, Code);{Get the number of nodes}
  Read(MapFile,InputString);      {in map}
  Readln(MapFile,InputString);
  DeleteSpaces(InputString);
  Val(InputString, FactMaxX, Code);{Get the Maximum X      }
  Read(MapFile,InputString);      {coordinate of factory floor}
  Readln(MapFile,InputString);
  DeleteSpaces(InputString);
  Val(InputString, FactMaxY, Code);{Get the Maximum Y      }
  Readln(MapFile);                {coordinate of factory floor}
  Readln(MapFile);
Repeat
  RowNr := RowNr +1;
  Read(MapFile,InputString); DeleteSpaces(InputString);
  Val(InputString, NodeNr[RowNr], Code);
  Read(MapFile,InputString); DeleteSpaces(InputString);
  Val(InputString, x[RowNr], Code);
  Read(MapFile,InputString); DeleteSpaces(InputString);
  Val(InputString, y[RowNr], Code);
  Read(MapFile,InputString); DeleteSpaces(InputString);
  Val(InputString, NrLinks[RowNr], Code);
  For LinkCount := 1 to NrLinks[RowNr] do
  Begin
    Read(MapFile,InputString);
    DeleteSpaces(InputString);

```

```

    Val(InputString, LinkNodeNr[RowNr,LinkCount,true], Code);
    Val(InputString, LinkNodeNr[RowNr,LinkCount,false],Code);
end;
Readln(MapFile);
Until RowNr = MaxNodeNr;
3 Close(MapFile);
end;{ Procedure GetMapInfo}

{*****}
Procedure InitMapWindow(FactMaxY , FactMaxX, MaxNodeNr:
    Integer; Title: String);

Var
    rt, LOWalarm, HIalarm, SetPnt      : RealType;
    Count, NrTraces, Grid, xdecs, ydecs : Integer;
    ratchf                             : BOOLEAN;
    XYTags                             : TagArrayType;
    Xunits, Yunits                     : TagType;
    LineColor, LineFill                : rtintarraytype;
    MapFilePath                        : string;

Begin    {Position window}
    rtsetpercentwindow(rtstat[5], 0, 0.15, 1, 1);
    rt := 0.99;  NrTraces := MaxNodeNr+2;    grid := 0;
    ratchf := False; xdecs := 0;    ydecs := 0;
    For Count := 0 to NrTraces do XYTags[Count] := "";
    YUnits := 'X position [cm]';
    XUnits := 'Y position [cm]';
    For Count := 0 to NrTraces do
        Begin
            LineColor[Count] := 10;
            LineFill[Count] := EmptyFill;
        End;
    rtInitWindowColors(rtstat[5], DARKGRAY, RED, LIGHTBLUE,
        LIGHTRED, YELLOW, WHITE, WHITE);
    rtsetwindowoptions(rtstat[5], RT_BORDER);

```

```

rtsetupxygraph(rtstat[5], FactoryFloorminX, FactMaxX,
               FactoryFloorminY, FactMaxY, 1000, rt,
               NrTraces, grid, LOWalarm, HIalarm, setpnt,
               xdecs, ydecs, title, Xunits, Yunits,
               XYTags, LineColor, LineFill, ratchf);

```

```

end;{of Procedure InitMapWindow}

```

```

{*****}

```

```

Procedure DeleteSpaces(var InputString:FileString);

```

```

Var

```

```

Count : Integer;

```

```

InChar : String[1];

```

```

Begin

```

```

    Count := 1;

```

```

    Repeat

```

```

        InChar := Copy(InputString,Count, 1);

```

```

        Count := Count +1;

```

```

    Until (InChar = ' ') or (count > 10);

```

```

    If count < 9 then {because a filestring has 8 characters}

```

```

        Delete(InputString,Count-1,10-Count);

```

```

End;{of Procedure DeleteSpaces}

```

```

{*****}

```

```

Procedure DisplayNodes(MaxNodeNr : Integer);

```

```

Const

```

```

    DrawOffset = 6;

```

```

Var

```

```

    Gd, Gm          : integer;

```

```

    Xratio, Yratio   : Real;

```

```

    RowNr, XYgraphCount : Integer;

```

```

Begin

```

```

    XYgraphCount:=0;

```



```

For RowNr := 0 to MaxNodeNr do
Begin
  Pos[XYgraphCount] := x[RowNr]-DrawOffset;
  Pos[XYgraphCount+1] := y[RowNr]-DrawOffset;
  XYgraphCount := XYgraphCount+2;
end;
Pos[MaxNodeNr+MaxNodeNr+2] := x[StartNode];
Pos[MaxNodeNr+MaxNodeNr+3] := y[StartNode];
rtupdatexygraph(rtstat[5],Pos);
For XYgraphCount := 0 to (MaxNodeNr+MaxNodeNr+1) do
Begin
  Pos[XYgraphCount+1] := Pos[XYgraphCount+1]+2*DrawOffset;
  XYgraphCount := XYgraphCount+1;
end;
rtupdatexygraph(rtstat[5],Pos);
For XYgraphCount := 0 to (MaxNodeNr+MaxNodeNr+1) do
Begin
  Pos[XYgraphCount] := Pos[XYgraphCount] + 2*DrawOffset;
  XYgraphCount := XYgraphCount+1;
end;
rtupdatexygraph(rtstat[5],Pos);
For XYgraphCount := 0 to (MaxNodeNr+MaxNodeNr+1) do
Begin
  Pos[XYgraphCount+1] := Pos[XYgraphCount+1]-2*DrawOffset;
  XYgraphCount := XYgraphCount+1;
end;
rtupdatexygraph(rtstat[5],Pos);
For XYgraphCount := 0 to (MaxNodeNr+MaxNodeNr+1) do
Begin
  Pos[XYgraphCount] := Pos[XYgraphCount] - 2*DrawOffset;
  XYgraphCount := XYgraphCount+1;
end;
rtupdatexygraph(rtstat[5],Pos);
end;{of Procedure DisplayNodes}

{*****}

```

```
Procedure DisplayLinks(MaxNodeNr : Integer);
```

```
Var
```

```
xyGraphCount, RowNr, LinkCount : Integer;
```

```
3 Begin
```

```
xyGraphCount := 0;
```

```
For RowNr := 0 to MaxNodeNr do
```

```
Begin
```

```
For LinkCount := 1 to NrLinks[RowNr] do
```

```
Begin
```

```
Pos[xyGraphCount] := x[RowNr];
```

```
Pos[xyGraphCount+1] := y[RowNr];
```

```
rtupdatexygraph(rtstat[5],Pos);
```

```
Pos[xyGraphCount]:= x[LinkNodeNr[RowNr,LinkCount,true]];

```

```
Pos[xyGraphCount+1]:=
```

```
    y[LinkNodeNr[RowNr,LinkCount,true]];

```

```
rtupdatexygraph(rtstat[5],Pos);
```

```
Pos[xyGraphCount] := x[RowNr];
```

```
Pos[xyGraphCount+1] := y[RowNr];
```

```
rtupdatexygraph(rtstat[5],Pos);
```

```
end;
```

```
xyGraphCount := xyGraphCount + 2;
```

```
end;
```

```
end;{of Procedure DisplayLinks}
```

```
{*****}
```

```
Procedure SaveStartPosition(int:integer; var instring:string);
```

```
Var
```

```
code : integer;
```

```
Begin
```

```
Case (int) of
```

```
0: val(instring, xpos,code);
```

```
1: val(instring, ypos,code);
```

```

2: val(instring, CurrentHeadAngle,code);
3: val(instring, StartNode,code);
4: val(instring, GoalNodeNr,code);
end;
end;{of Procedure SaveStartPosition}

```

```

{*****}

```

```

Procedure InputStartPosition(var X,Y, CHA: Real;
                             FactMaxY, FactMaxX, MaxNodeNr : Integer;
                             var Error : Boolean);

```

Type

```

ptagtype = Array[0..4] OF _string40;
ptagpntr = ^ptagtype;

```

Var

```

tags, fieldstrings      : ptagpntr;
s                        : string[4];
i, nt                   : INTEGER;
itemjust, fieldw        : Array[0..15] OF Integer;
boxcoords               : Array[0..5] OF pnt2d;
dbt                     : dialogframepntr;
gdcolors                : Array[0..5] OF Integer;
MapFilePath, Xstring,
Ystring, CHAstring      : String;

```

Begin

```

gdcolors[0] := BLACK;  gdcolors[1] := WHITE;
gdcolors[2] := WHITE;  gdcolors[3] := LIGHTGRAY;
nt := 5;
NEW(fieldstrings);
NEW(tags);
For i:=0 to nt-1 do
Begin
  boxcoords[i].x := 28;
  boxcoords[i].y := 2 + 3*i;

```



```

itemjust[i] := 2;
fieldw[i] := 5;
end;
str(round(x), Xstring);
str(round(y), Ystring);
str(round(CHA), CHAstring);
fieldstrings^[0] := {'100'} Xstring;
tags^[0] := 'X coordinate [cm]';
fieldstrings^[1] := {'100'} Ystring;
tags^[1] := 'Y coordinate [cm]';
fieldstrings^[2] := {'90'} CHAstring;
tags^[2] := 'Current Orientation';
fieldstrings^[3] := '0';
tags^[3] := 'Nearest Node';
fieldstrings^[4] := '10';
tags^[4] := 'Goal Node';
dbt := qcopendialogframe(100, 50, 35, 28, @gdcolors,
    '=====', DLG_OUTLINE or DLG_LMOVE or
    DLG_3DCONTROLS or DLG_OKButton or DLG_CancelButton,
    NilDFCloseProc, NilDFMoreFunc, 'help');
qcopengeneraldialogbox(dbt, 1, 1, 32, 21, @tags^,
    @fieldstrings^, @fieldw, @boxcoords, @itemjust,
    1, nt, 'MOBROB Start Position', DLG_OUTLINE,
    NilGDProcessFunc, SaveStartPosition);
qcdrawdialogframe(dbt);
i := qcprocessdialogframe(dbt);
qcclosedialogframe(dbt);
Dispose(tags);
Dispose(fieldstrings);
x := Xpos; {these gobal variables are changed
            by the function SaveStartPosition}
y := Ypos;
CHA := CurrentHeadAngle;
Error := False;
If (StartNode < 0) or (GoalNodeNr < 0)
    or (StartNode > MaxNodeNr) or (GoalNodeNr > MaxNodeNr)

```

```

    or (x < 0) or (x > FactMaxX) or (y < 0) or (y > FactMaxY)
    or (CHA < -360) or (CHA > 360)
    then Error := true;
end;{of Procedure InputStartPosition}

```

```

3 {*****}

```

```

Procedure SaveCurrentPosition(int: integer;
                               var instring: string);

```

```

Var
code : integer;

```

```

Begin
Case (int) of
0: val(instring, xpos,code);
1: val(instring, ypos,code);
2: val(instring, CurrentHeadAngle,code);
end;
end;{of SaveCurrentPosition}

```

```

Procedure SaveAngles(int: integer; var instring: string);

```

```

Var
code : integer;

```

```

Begin
Case (int) of
0: val(instring, scenario,code);
1: val(instring, alpha,code);
2: val(instring, Beta,code);
3: val(instring, Mu,code);
end;
end;{of SaveCurrentPosition}

```

```

{*****}

```

```

Procedure GetCurrentPos(Var X, Y, CHA: real;maxx, maxy : real);

```

Var

Xstring, Ystring, CHAstring : String;

Type

```

3  ptagtype = Array[0..5] of _string40;
   ptagpntr = ^ptagtype;

```

Var

```

tags, fieldstrings      : ptagpntr;
s                        : string[4];
i, nt                   : Integer;
itemjust, fieldw        : Array[0..15] of Integer;
boxcoords               : Array[0..5] of pnt2d;
dbt                     : dialogframepntr;
gdcolors                : Array[0..5] of Integer;
error                   : boolean;

```

Begin

If NavigateHelp <> 1 then

Begin

```

gdcolors[0] := BLACK;  gdcolors[1] := WHITE;
gdcolors[2] := WHITE;  gdcolors[3] := LIGHTGRAY;
nt := 3;
NEW(fieldstrings);
NEW(tags);
For i:=0 to nt-1 do
Begin
  boxcoords[i].x := 28;
  boxcoords[i].y := 2 + 3*i;
  itemjust[i] := 2;
  fieldw[i] := 5;
end;
str(X:0:0,Xstring);
str(Y:0:0,Ystring);
str(CHA:0:0,CHAstring);

```



```

fieldstrings^[0] := Xstring;
tags^[0] := 'X coordinate [cm]';
fieldstrings^[1] := Ystring;
tags^[1] := 'Y coordinate [cm]';
fieldstrings^[2] := CHAstring;
tags^[2] := 'Current Orientation';
dbt := qcopendialogframe(0, 240, 35, 28, @gdcolors,
    '=====', DLG_OUTLINE or DLG_LMOVE or
    DLG_3DCONTROLS or DLG_OKButton,
    NilDFCloseProc, NilDFMoreFunc, 'help');
qcopengeneraldialogbox(dbt, 1, 1, 32, 21, @tags^,
    @fieldstrings^, @fieldw, @boxcoords,
    @itemjust, 1, nt, 'Enter Positioning Angles',
    DLG_OUTLINE, NilGDProcessFunc,
    SaveCurrentPosition);
qcdrawdiallogframe(dbt);
i := qcprocessdialogframe(dbt);
qcclosedialogframe(dbt);
Dispose(tags);
Dispose(fieldstrings);
end else
begin
    gdcolors[0] := BLACK;    gdcolors[1] := WHITE;
    gdcolors[2] := WHITE;    gdcolors[3] := LIGHTGRAY;
    nt := 4;
    NEW(fieldstrings);
    NEW(tags);
    For i:=0 to nt-1 do
    Begin
        boxcoords[i].x := 28;
        boxcoords[i].y := 2 + 3*i;
        itemjust[i] := 2;
        fieldw[i] := 5;
    end;
    fieldstrings^[0] := "";
    tags^[0] := 'Scenario [0..3]';

```

```

fieldstrings^[1] := "";
tags^[1] := 'Alpha in Degrees';
fieldstrings^[2] := "";
tags^[2] := 'Beta in Degrees';
fieldstrings^[3] := "";
tags^[3] := 'Mu in Degrees';
dbt := qcopendialogframe(0, 240, 35, 28, @gdcolors,
    '=====', DLG_OUTLINE or DLG_LMOVE or
    DLG_3DCONTROLS or DLG_OKButton,
    NilDFCloseProc, NilDFMoreFunc, 'help');
qcopengeneraldialogbox(dbt, 1, 1, 32, 21, @tags^,
    @fieldstrings^, @fieldw, @boxcoords,
    @itemjust, 1, nt, 'Enter Current Position',
    DLG_OUTLINE, NilGDProcessFunc,
    SaveAngles);
qcdrawdialogframe(dbt);
i := qcprocessdialogframe(dbt);
qcclosedialogframe(dbt);
Dispose(tags);
Dispose(fieldstrings);

{AutoNavHelp(alpha1,beta1,mu,scenario);}
AutomaticUpdate(maxx, maxy, alpha, beta, mu, scenario, Xpos, Ypos, CurrentHeadAngle);
end;
x := Xpos;
y := Ypos;
CHA := CurrentHeadAngle;
end;{of Procedure GetCurrentPos}

{*****}
Procedure DoPathPlanning(MapFilePath: String;
    var NodePath: String);

Const
    Infinity    = 1e10;
    RoundingError = 0.5;

```

```

Var
  NodeAssigned          : Boolean;
  PathFile              : text;
  FactMaxY , FactMaxX, MaxNodeNr   : Integer;
  MinDist              : real;
  NodeNrString          : array[0..NodeLimit]
                        of String[2];
  NodeCounter, ReverseLinkCount,
  TempListCount, CurrentNodeNr,
  MinDistNodeNr, NodeNr, FromNodeNr,
  ToNodeNr, LinkCount   : Integer;
  PermList, TempList    : array[0..NodeLimit]
                        of real;
  Distance              : array[0..NodeLimit,0..NodeLimit] of real;

Begin
  GetMapInfo(MapFilePath, FactMaxY , FactMaxX, MaxNodeNr);
  {****Calculates the shortest distance between all nodes****}
  For NodeNr := 0 to MaxNodeNr do
    Begin
      For LinkCount := 1 to NrLinks[NodeNr] do
        Begin
          ToNodeNr := LinkNodeNr[NodeNr,LinkCount,true];
          FromNodeNr := NodeNr;
          Distance[FromNodeNr, ToNodeNr] :=
            Sqrt(sqr(x[ToNodeNr]-x[FromNodeNr]) +
                sqr(y[ToNodeNr]-y[FromNodeNr]))
        end;
      end;
    end;
  For NodeNr := 0 to NodeLimit do
    Begin
      PermList[NodeNr] := Infinity;
      TempList[NodeNr] := Infinity;
    end;
  {***Constructs permanent and temporary list according to****}

```



```

{***Dijkstra's algorithm for shortest path in networks****}
PermList[Startnode] := 0;
  {Distance to itself = 0 => first entry in Permanent list}
CurrentNodeNr := StartNode;
Repeat
  For LinkCount := 1 to NrLinks[CurrentNodeNr] do
    Begin
      If LinkNodeNr[CurrentNodeNr,LinkCount,True] =
        LinkNodeNr[CurrentNodeNr,LinkCount,False] then
        Begin
          Templist[LinkNodeNr[CurrentNodeNr,LinkCount,true]] :=
            MinOf2(Distance[CurrentNodeNr,
              LinkNodeNr[CurrentNodeNr,LinkCount,true]] +
              PermList[CurrentNodeNr],
            Templist[LinkNodeNr[CurrentNodeNr,LinkCount,true]]);
          ReverseLinkCount:=0;
          Repeat {remove the reverse links of current link}
            ReverseLinkCount := ReverseLinkCount+1;
          until LinkNodeNr[LinkNodeNr[CurrentNodeNr,
            LinkCount,true],ReverseLinkCount,true]
            = CurrentNodeNr;
          LinkNodeNr[LinkNodeNr[CurrentNodeNr,LinkCount,true],
            ReverseLinkCount,False] := LinkLimit ;
        end;
      end;
    MinDist := 1e20;
    For TempListCount := 0 to NodeLimit do
      Begin {Selects the smallest entry in temporary list}
        If TempList[TempListCount] < Mindist Then
          Begin
            MinDist := TempList[TempListCount] ;
            MinDistNodeNr := TempListCount;
          end;
        end;
      PermList[MinDistNodeNr] := MinDist; {Make Min permanent}
      TempList[MinDistNodeNr] := Infinity;

```

```

        {Remove min from templist}
    CurrentNodeNr := MinDistNodeNr;
    Until MinDistNodeNr = GoalNodeNr;
    {*****Work backwards to find shortest path*****}
    CurrentNodeNr := GoalNodeNr;
    NodeCounter := 0;
    Repeat
    NodeAssigned := False;
    For LinkCount := 1 to NrLinks[CurrentNodeNr] do
    Begin
        If (Distance[CurrentNodeNr,
            LinkNodeNr[CurrentNodeNr,LinkCount,true]]
            <= (PermList[CurrentNodeNr] + RoundingError -
            PermList[LinkNodeNr[CurrentNodeNr,LinkCount,true]]))
        and
            (Distance[CurrentNodeNr,
            LinkNodeNr[CurrentNodeNr,LinkCount,true]]
            >= (PermList[CurrentNodeNr] - RoundingError -
            PermList[LinkNodeNr[CurrentNodeNr,LinkCount,true]]))
        then
        Begin
            If NodeAssigned = False then
            Begin
                Str(CurrentNodeNr,NodeNrString[NodeCounter]);
                NodeCounter := NodeCounter + 1;
                CurrentNodeNr := LinkNodeNr[CurrentNodeNr,
                    LinkCount,true];
            end;
            NodeAssigned := True;
        end;
    end;
    Until CurrentNodeNr = StartNode;
    Str(CurrentNodeNr,NodeNrString[NodeCounter]);
    Delete(MapFilePath,Length(MapFilePath)-2,3);
    NodePath := MapFilePath; {Create path file with the same }
    Insert('PTH',NodePath,Length(NodePath)+1); {name as mapfile}

```

```

Assign(PathFile,NodePath); {but with a .PTH file extension}
Rewrite(PathFile);
For NodeCounter := NodeCounter downto 0 do
Begin
  Writeln(PathFile,NodeNrString[NodeCounter]);
end;
Writeln(PathFile,'END');
Close(PathFile);
end;{of Procedure DoPathPlanning}
{*****}
Procedure GetNodeCoords(NthNode : integer; Var Xpos,Ypos:real);
Begin
  Xpos := x[NthNode];
  Ypos := y[NthNode];
end;{of Procedure GetNodeCoords}
{*****}
Begin
end.

```

UNIT NAVDRIVE

```

Unit NavDrive;
{*****}
Interface
{*****}
Uses
  Crt;
VAR
  x, y, Orientation: Real;
{*****}
Procedure CalcPosition(alpha,beta,mu,maxx,maxy : real;scenario:integer);
Procedure AutomaticUpdate(maxx, maxy, alpha1, beta1, mu1 : real;
  scenario1 : integer;
  Var Xpos, Ypos, CurrentHeadAngle :real);
{*****}
Implementation
{*****}

```



```

Procedure CalcPosition(alpha,beta,mu,maxx,maxy : real;scenario:integer);
{Determine Tan Gamma for a specific scenario}
{*****}
var
  CotAlpha, r, TanGamma, TanPow2, aplusb: real;
  c, m, psi, gamma : real;
  W : Real; {Angle between Mobrob's position and Beacon B's position}
  P : Real; {Angle between Mobrob's position and Beacon A's position}
  a,b : array[0..3] of real;
  pi : real;
Begin
  pi := 3.141592654;
  a[0] := maxy;
  a[1] := maxx;
  a[2] := maxy;
  a[3] := maxx;
  b[0] := maxx;
  b[1] := maxy;
  b[2] := maxx;
  b[3] := maxy;
  aplusb:= (Alpha+Beta);
  r := b[Scenario]*(cos(apusb)*sin(Alpha));
  TanGamma:=r/(b[Scenario]*sin(apusb)*sin(Alpha)-a[Scenario]*sin(Beta));
  Gamma:=ArcTan(TanGamma);
  CotAlpha:=cos(Alpha)/sin(Alpha);
  TanPow2:=(TanGamma)*(TanGamma);
  {*****}
  {Determine x and y co-ordinates}
  {*****}
  Case Scenario of
    0:begin
      x := (a[Scenario]*(1 + CotAlpha*TanGamma)*TanGamma)/(1+TanPow2);
      y := (a[Scenario]*(TanGamma-CotAlpha)*TanGamma)/(1+TanPow2);
    end; {0}
    1:begin
      x := a[Scenario]-(a[Scenario]*(TanGamma-CotAlpha)*TanGamma)/(1+TanPow2);

```

```

    y := (a[Scenario]*(1+CotAlpha*TanGamma)*TanGamma)/(1+TanPow2);
end; {1}
2:begin
    x := b[Scenario]-(a[Scenario]*(1+CotAlpha*TanGamma)*TanGamma)/(1+TanPow2);
    y := a[Scenario]-(a[Scenario]*(TanGamma-CotAlpha)*TanGamma)/(1+TanPow2);
end; {2}
3:begin
    x := (a[Scenario]*(TanGamma-CotAlpha)*TanGamma)/(1+TanPow2);
    y := b[Scenario] - (a[Scenario]*(1+CotAlpha*TanGamma)*TanGamma)/(1+TanPow2);
end; {3}
End; {Case Scenario}
x := x;
y := y;
{*****}
{Calculation of direction the robot is looking in}
{*****}
W := abs(arctan((x)/(y)));
P := abs(arctan((x)/(y-maxy)));
Case Scenario OF
0: Orientation := (-Pi + mu + Alpha + W)*180/Pi;
1: Orientation := (-Pi + mu + W)*180/Pi;
2: Orientation := (-2*Pi + Beta + Alpha + mu - P)*180/Pi;
3: Orientation := (Alpha + mu - P)*180/Pi;
End;{case Statement}
End;{of Procedure CalcPosition}
{*****}
Procedure AutomaticUpdate(maxx, maxy, alpha1, beta1, mu1 : real;
    scenario1 : integer;
    Var Xpos, Ypos, CurrentHeadAngle :real);
var
    pi : real;
Begin
    pi := 3.141592654;
    alpha1 := alpha1*pi/180;
    beta1 := beta1*pi/180;
    mu1 := mu1*pi/180;

```

```

scenario1 := scenario1;
CalcPosition(alpha1,beta1,mu1,maxx,maxy,scenario1);
Xpos := x;
Ypos := y;
CurrentHeadAngle := Orientation;
end;
END.

```

UNIT RFU309

```

unit RFU309;    { Weergawe 3.09 }
{ program om Serie data met RF te versend }

```

Interface

uses

crt;

const

arraygrootte = 100;

type

ArrayType= array[1..arraygrootte] of integer;

procedure Init(z : integer);

procedure WysLer(LerNaam : string);

procedure OntvangLer(LerNaam : string);

procedure StuurLer(LerNaam : string);

procedure LerNaamInlees(var LerNaam : string);

procedure OntvangArray(var ArrayOntvang : ArrayType);

procedure StuurArray(ArrayStuur : ArrayType);

procedure Enkodeer(r1,r2,r3,r4,r5,r6,r7,r8 : real;

var ArrayStuur : ArrayType);

procedure Dekodeer(var r1,r2,r3,r4,r5,r6,r7,r8 : real;

ArrayOntvang : ArrayType);

Implementation

const

```

sbase      = $3F8; { Basis adres vir seriepoort }
LCR        = sbase + 3; { Line Control Register }
IER        = sbase + 1; { Interrupt Enable Register }
DLMSB      = sbase + 1; { Divisor Latch MSB }
DLLSB      = sbase; { Divisor Latch LSB }
LSR        = sbase + 5; { Line Status Register }
k_grootte_const = 10000;

```

type

```

datatype = array[1..k_grootte_const] of char;

```

var

```

kies      : char;
lernaam   : string;

```

procedure Init(z : integer);

```

{ Inisialiseer serie poort en stel RF module vir ontvang of stuur }

```

begin

```

port[$378] := z;
port[LCR]  := $87; { 8 bit, geen pariteit, twee stoppe }
port[DLLSB] := $0C; { Baud Rate }
port[DLMSB] := 0; { Baud Rate }
port[LCR]  := 7;
port[IER]  := 0; { Interrupts disabled }
port[LSR]  := 0;

```

```
end; {Init }
```

```
procedure WysLer(LerNaam : string);
```

```
{ Vertoon 'n le^r op die skerm }
```

```
var
```

```
  k      : char;
```

```
  lsrdata : integer;
```

```
  inler   : file of char;
```

```
begin
```

```
  clrscr;
```

```
  assign(inler,LerNaam);
```

```
  reset(inler);
```

```
  while not eof(inler) do
```

```
    begin
```

```
      read(inler,k);
```

```
      write(k);
```

```
    end;
```

```
  close(inler);
```

```
  readln;
```

```
end; { procedure WysLer }
```

```
procedure OntvangLer(LerNaam : string);
```

```
{ Ontvang 'n teks le^r vanaf serie-poort }
```

```
var lsrdata, x, xx : integer;
```

```
  k      : datatype;
```

```
  voort, stop : boolean;
```

```
  uitler      : file of char;
```

```
begin
```

```

Init(2);
x:=3;
voort:=false;
stop:=false;
repeat
  lsrdata := port[LSR];
  if (lsrdata AND 1)=1 then { toets of daar data in buffer is }
  begin
    xx := port[sbase];
    if (xx = 10) then begin
      if k[x-1] <> char(13) then begin
        xx := 13;
        k[x] := char(xx);
        inc(x);
        xx := 10;
      end;
    end;
    k[x]:=char(xx);
    port[LSR] := 0;
    if (k[x-2]='S') and (k[x-1]='L') and (k[x]='M') then
      begin
        stop:=true;
        x:=x-3;
      end;
    if not voort then
      if (k[x-2]='M') and (k[x-1]='L') and (k[x]='S') then
        begin
          voort:=true;
          x:=0;
        end
      else if (k[x-1]<>'M') and (k[x]<>'M') then
        x:=2;
      inc(x);
    end;
  until stop;

```



```

assign(uitler,LerNaam);
rewrite(uitler);
for xx:=1 to (x-1) do
  write(uitler,k[xx]);
close(uitler);
end; { procedure OntvangLer }

```

```

procedure StuurLer(LerNaam : string);
{ Lees le^r en stuur na serie poort }

```

```

var
  k           : datatype;
  voort       : boolean;
  x, lsrdata, k_grootte, k_grootte2 : integer;
  inler       : file of char;

```

```

begin
  Init(5);
  k_grootte := k_grootte_const;

```

```

assign(inler,LerNaam);
reset(inler);
voort:=true;
for x:=1 to 799 do
  k[x]:='U';
for x := 800 to 1000 do
  k[x]:='m';
k[1001]:='M';
k[1002]:='L';
k[1003]:='S';
for x:=1004 to k_grootte do
  if not eof(inler) then
    read(inler,k[x])
  else if eof(inler) then

```

```

begin
  k_grootte2:=x;
  x:=k_grootte;
end; { for x:=1 to }
close(inler);

k[k_grootte2+1]:='S';
k[k_grootte2+2]:='L';
k[k_grootte2+3]:='M';

for x:= 1 to (k_grootte2+3) do
begin
  voort:=true;
  while voort do
  begin
    lsldata := port[LSR];
    if (lsldata and 32)=32 then
    begin
      port[sbase] := ord(k[x]);
      voort:=false;
    end;
  end; { while voort }
end; { for x := }
end; { procedure StuurLer }

```

```

procedure LerNaamInlees(var LerNaam : string);

```

```

begin
  write('Tik die naam van die l`er in : ');
  readln(LerNaam);
end; { procedure LerNaamInlees }

```

```

procedure OntvangArray(var ArrayOntvang : ArrayTipe);

var lsrdata, x, xx : integer;
    k      : array[1..(arraygrootte+8)] of integer;
    voort, stop : boolean;

begin
    Init(2);
    x:=3;

    voort:=false;
    stop:=false;
    repeat
        lsrdata := port[LSR];
        if (lsrdata AND 1)=1 then { toets of daar data in buffer is }
            begin
                k[x] := portw[sbase];
                port[LSR] := 0;
                if not voort then
                    if (char(k[x-2])='M') and (char(k[x-1])='L') and (char(k[x])='S') then
                        begin
                            voort:=true;
                            x:=0;
                        end;
                    inc(x);
                end;
            until (x=arraygrootte+1);

        for x:=1 to (arraygrootte) do
            begin
                arrayontvang[x]:=k[x];
            end;
        end; { procedure OntvangArray }

```



```

procedure StuurArray(ArrayStuur : ArrayTipe);

var
  k          : Array[1..(arraygrootte+8)] of integer;
  voort      : boolean;
  x, lsldata : integer;
  inler      : file of char;

begin
  Init(5);
  voort:=true;
  for x:=1 to 5 do
    k[x]:=ord('m');
    k[6]:=ord('M');
    k[7]:=ord('L');
    k[8]:=ord('S');
  for x:=9 to (arraygrootte+8) do
    k[x] := ArrayStuur[x-8];

  for x:= 1 to (arraygrootte+8) do
    begin
      voort:=true;
      while voort do
        begin
          lsldata := port[LSR];
          if (lsldata and 32)=32 then
            begin
              portw[sbase] := k[x];
              voort:=false;
            end;
          end; { while voort }
        end; { for x := }
      end; { procedure StuurArray }

```

```
procedure Enkodeer(r1,r2,r3,r4,r5,r6,r7,r8 : real; var ArrayStuur : ArrayType);
```

```
  var
```

```
    r    : array[0..7] of longint;
```

```
    x,y,l,c : integer;
```

```
    s     : string[12];
```

```
    k     : string[1];
```

```
begin
```

```
  r[0] := round(r1*10);
```

```
  r[1] := round(r2*10);
```

```
  r[2] := round(r3*10);
```

```
  r[3] := round(r4*10);
```

```
  r[4] := round(r5*10);
```

```
  r[5] := round(r6*10);
```

```
  r[6] := round(r7*10);
```

```
  r[7] := round(r8*10);
```

```
  for x := 0 to 7 do
```

```
    begin
```

```
      if r[x] < 0 then
```

```
        ArrayStuur[x*12+12]:=0
```

```
      else ArrayStuur[x*12+12]:=1;
```

```
      r[x]:=abs(r[x]);
```

```
      str(r[x],s);
```

```
      l:=length(s);
```

```
      for y := 1 to l do
```

```
        begin
```

```
          k := copy(s,y,1);
```

```
          val(k,ArrayStuur[x*12+y],c);
```

```
        end;
```

```
      for y := (l+1) to 11 do
```

```

    ArrayStuur[x*12+y]:=120;
end;
end; { procedure Enkodeer }

```

```

procedure Dekodeer(var r1,r2,r3,r4,r5,r6,r7,r8 : real; ArrayOntvang : ArrayTipe);
var
    r_array : array[0..7] of real;
    s      : array[0..7] of string[12];
    x,y,c,i : integer;
    st     : string[12];
begin
    for x := 0 to 7 do
        begin
            s[x]:="";
            for y:=1 to 11 do
                begin
                    if ArrayOntvang[x*12+y]=120 then
                        y:=11
                    else
                        begin
                            Str(ArrayOntvang[x*12+y]:0,st);
                            s[x]:=s[x]+st;
                        end;
                    end;
                end
            val(s[x],r_array[x],c);

            if ArrayOntvang[x*12+12]=0 then r_array[x]:=r_array[x]/(-10)
            else r_array[x]:=r_array[x]/10;
        end;

    r1 := r_array[0];
    r2 := r_array[1];
    r3 := r_array[2];
    r4 := r_array[3];

```



```
r5 := r_array[4];  
r6 := r_array[5];  
r7 := r_array[6];  
r8 := r_array[7];  
end; { procedure Dekodeer }  
3 end. { program }
```

APPENDIX E: MOBROB DELPHI PROGRAMS

The Delphi code created is included in this appendix.

FILESEND.PAS

unit Filesend;

{ \$X+ }

interface

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
StdCtrls, ComCtrls, VaConst, VaTypes, VaClasses, VaComm, ExtCtrls, Menus;

type

TfrmMain = class(TForm)
 VaComm1: TVaComm;
 StatusBar1: TStatusBar;
 Panel1: TPanel;
 ButtonTransmit: TButton;
 ButtonQTransmit: TButton;
 Panel2: TPanel;
 Panel3: TPanel;
 Memo2: TMemo;
 Panel4: TPanel;
 Memo1: TMemo;
 Splitter1: TSplitter;
 Panel5: TPanel;
 ButtonReset: TButton;
 OpenDialog1: TOpenDialog;
 QPathPlanning: TButton;
 PathPlanning: TButton;

```

MainMenu1: TMainMenu;
About1: TMenuItem;
Image1: TImage;
procedure FormCreate(Sender: TObject);
procedure ButtonResetClick(Sender: TObject);
3 procedure ButtonTransmitClick(Sender: TObject);
procedure Comm1TxEmpty(Sender: TObject);
procedure Comm1Break(Sender: TObject);
procedure Comm1Cts(Sender: TObject);
procedure Comm1Dsr(Sender: TObject);
procedure Comm1Error(Sender: TObject; Errors: Integer);
procedure Comm1Ring(Sender: TObject);
procedure Comm1Rlsd(Sender: TObject);
procedure ButtonQTransmitClick(Sender: TObject);
procedure VaComm1Data(Sender: TObject; Count: Integer);
procedure VaComm1Event(Sender: TObject);
procedure VaComm1Open(Sender: TObject);
procedure VaComm1Close(Sender: TObject);
procedure QPathPlanningClick(Sender: TObject);
procedure PathPlanningClick(Sender: TObject);
procedure About1Click(Sender: TObject);
private
  procedure HandleException(Sender: TObject; E: Exception);
public
  { Public declarations }
end;

var
  frmMain: TfrmMain;

implementation

uses about, ReminderSend, ReminderReceive;

{$R *.DFM}

```



```
procedure TfrmMain.FormCreate(Sender: TObject);
```

```
begin
```

```
  Application.OnException := HandleException;
```

```
  VaComm1.PortNum := 1;
```

```
  VaComm1.BaudRate := br9600;           {Baudrate = 9 600}
```

```
  VaComm1.Databits := db8;              {Databits = 8}
```

```
  VaComm1.Parity := paNone;             {Parity = None}
```

```
  VaComm1.StopBits := sb1;              {Stopbits = 10}
```

```
end;
```

```
procedure TfrmMain.HandleException(Sender: TObject; E: Exception);
```

```
begin
```

```
  if E is EVaCommError then
```

```
    with E as EVaCommError do
```

```
      ShowMessage(Message);
```

```
end;
```

```
procedure TfrmMain.ButtonResetClick(Sender: TObject);
```

```
begin
```

```
  Memo1.Lines.Clear;
```

```
  Memo2.Lines.Clear;
```

```
end;
```

```
procedure TfrmMain.ButtonTransmitClick(Sender: TObject);
```

```
var
```

```
  I: Integer;
```

```
  S, Filename: String;
```

```
  ch, filerec, FilenameCH, SCH : array[1..100000] of char;
```

```
  F1 : Textfile;
```

```
  counter, counter2 : integer;
```

```
  buffer : char;
```

```
  received : boolean;
```

```
  test, ok : boolean;
```

```
  finished : boolean;
```

```

filesize : integer;
filesize1 : integer;
filesize2 : integer;

begin
  RemSend.ShowModal;

  Opendialog1.Execute;
  S := Opendialog1.FileName;

  VaComm1.PortNum := 1;
  VaComm1.BaudRate := br9600;           {Baudrate = 9 600}
  VaComm1.Databits := db8;              {Databits = 8}
  VaComm1.Parity := paNone;             {Parity = None}
  VaComm1.StopBits := sb1;              {Stopbits = 10}
  VaComm1.Open;                         {Open the comm port}
  Comm1Cts(VaComm1);
  Comm1Dsr(VaComm1);
  Comm1Ring(VaComm1);
  Comm1Rltd(VaComm1);
  for i := 1 to 995 do begin
    ch[i] := 'U';
  end;
  for i := 996 to 1000 do begin
    ch[i] := 'm';
  end;
  ch[1001] := 'M';
  ch[1002] := 'L';
  ch[1003] := 'S';
  i := 1003;

  AssignFile(F1, 'c:\Mobrob Progs\Lydia Progs\Matlab Progs\current.pos');
  Reset(F1);
  while not Eof(F1) do begin
    inc(i);
    Read(F1, Ch[i]);
  end;
end;

```

end;

CloseFile(F1);

inc(i);

Ch[i] := 'S';

Ch[i+1] := 'L';

Ch[i+2] := 'M';

filesize := i + 2;

for i := 1 to filesize do begin

OK := VaComm1.WriteText(ch[i]);

if not OK then

Memo1.Lines.add('Error writing');

end;

for i := 1 to 10000 do begin

counter := i;

end;

for i := 1 to 995 do begin

ch[i] := 'U';

end;

for i := 996 to 1000 do begin

ch[i] := 'm';

end;

ch[1001] := 'M';

ch[1002] := 'L';

ch[1003] := 'S';

i := 1003;

AssignFile(F1, S);

Reset(F1);


```

while not Eof(F1) do begin
    inc(i);
    Read(F1, Ch[i]);

end;

CloseFile(F1);

inc(i);
Ch[i] := 'S';
Ch[i+1] := 'L';
Ch[i+2] := 'M';

filesize := i + 2;

for i := 1 to filesize do begin
    OK := VaComm1.WriteText(ch[i]);
    if not OK then
        Memo1.Lines.add('Error writing');
end;

Counter := length(S);
Counter := Counter - 2;
Delete(S, Counter, 3);
Filename := S + 'pth';

for i := 1 to 10000 do begin
    counter := i;
end;

for i := 1 to 995 do begin
    ch[i] := 'U';
end;

for i := 996 to 1000 do begin
    ch[i] := 'm';
end;

```

```

ch[1001] := 'M';
ch[1002] := 'L';
ch[1003] := 'S';
i := 1003;

```

```

3 AssignFile(F1, Filename);
Reset(F1);
while not Eof(F1) do begin
    inc(i);
    Read(F1, Ch[i]);

```

```

end;

```

```

CloseFile(F1);

```

```

inc(i);
Ch[i] := 'S';
Ch[i+1] := 'L';
Ch[i+2] := 'M';

```

```

filesize := i + 2;

```

```

for i := 1 to filesize do begin
    OK := VaComm1.WriteText(ch[i]);
    if not OK then
        Memo1.Lines.add('Error writing');
end;

```

```

for i := 1 to 10000 do begin
    counter := i;
end;

```

```

VaComm1.Close;           {Close the comm port}

```

```
Comm1Cts(VaComm1);
Comm1Dsr(VaComm1);
Comm1Ring(VaComm1);
Comm1Rlsd(VaComm1);
```

```
3 RemReceive.ShowModal;
```

```
{VaComm1.Open;
Comm1Cts(VaComm1);
Comm1Dsr(VaComm1);
Comm1Ring(VaComm1);
Comm1Rlsd(VaComm1);
```

```
i:=3;
```

```
repeat
```

```
ch[1] := ch[2];
```

```
ch[2] := ch[3];
```

```
repeat
```

```
Ok := VaComm1.ReadChar(ch[i]);
```

```
until Ok
```

```
until (ch[i-2]='M') and (ch[i-1]='L') and (ch[i]='S');
```

```
i := 0;
```

```
repeat
```

```
inc(i);
```

```
repeat
```

```
Ok := VaComm1.ReadChar(ch[i]);
```

```
until Ok
```



```
until (ch[i-2]='S') and (ch[i-1]='L') and (ch[i]='M');
```

```
VaComm1.Close;
```

```
Comm1Cts(VaComm1);
```

```
Comm1Dsr(VaComm1);
```

```
Comm1Ring(VaComm1);
```

```
Comm1Rlsl(VaComm1);
```

```
filesize1:=i-3;
```

```
assignfile(F1,'c:\mobrob progs\lydia progs\matlab progs\current.pos');
```

```
rewrite(F1);
```

```
for Counter := 1 to filesize1-1 do begin
```

```
  write(F1,ch[counter]);
```

```
end;
```

```
closefile(F1); }
```

```
end;
```

```
procedure TfrmMain.Comm1TxEmpty(Sender: TObject);
```

```
begin
```

```
  Memo1.Lines.add("TxEmpty signal detected...");
```

```
end;
```

```
procedure TfrmMain.Comm1Break(Sender: TObject);
```

```
begin
```

```
  Memo1.Lines.add("Break signal detected...");
```

```
end;
```

```
procedure TfrmMain.Comm1Cts(Sender: TObject);
```

```
begin
```

```
  if VaComm1.CTS then
```

```
    StatusBar1.Panels[0].Text := 'CTS'
```

```
  else StatusBar1.Panels[0].Text := ";
```

```
end;
```

```
procedure TfrmMain.Comm1Dsr(Sender: TObject);
```

```

begin
  if VaComm1.DSR then
    StatusBar1.Panels[1].Text := 'DSR'
  else StatusBar1.Panels[1].Text := '';
end;

procedure TfrmMain.Comm1Ring(Sender: TObject);
begin
  if VaComm1.Ring then
    StatusBar1.Panels[2].Text := 'RING'
  else StatusBar1.Panels[2].Text := '';
end;

procedure TfrmMain.Comm1Rlsd(Sender: TObject);
begin
  if VaComm1.Rlsd then
    StatusBar1.Panels[3].Text := 'RLSD'
  else StatusBar1.Panels[3].Text := '';
end;

procedure TfrmMain.Comm1Error(Sender: TObject; Errors: Integer);
begin
  if (Errors and CE_BREAK > 0) then Memo1.Lines.add(sCE_BREAK);
  if (Errors and CE_DNS > 0) then Memo1.Lines.add(sCE_DNS);
  if (Errors and CE_FRAME > 0) then Memo1.Lines.add(sCE_FRAME);
  if (Errors and CE_IOE > 0) then Memo1.Lines.add(sCE_IOE);
  if (Errors and CE_MODE > 0) then Memo1.Lines.add(sCE_MODE);
  if (Errors and CE_OOP > 0) then Memo1.Lines.add(sCE_OOP);
  if (Errors and CE_OVERRUN > 0) then Memo1.Lines.add(sCE_OVERRUN);
  if (Errors and CE_PTO > 0) then Memo1.Lines.add(sCE_PTO);
  if (Errors and CE_RXOVER > 0) then Memo1.Lines.add(sCE_RXOVER);
  if (Errors and CE_RXPARITY > 0) then Memo1.Lines.add(sCE_RXPARITY);
  if (Errors and CE_TXFULL > 0) then Memo1.Lines.add(sCE_TXFULL);
end;

procedure TfrmMain.ButtonQTransmitClick(Sender: TObject);

```

```

var
  I: Integer;
  S: string;
  ch, filerec : array[1..100000] of char;
  F1 : Textfile;
3  counter, counter2 : integer;
  buffer : char;
  received : boolean;
  test, ok : boolean;
  finished : boolean;
  filesize : integer;
  filesize1 : integer;
  filesize2 : integer;

begin
  RemSend.ShowModal;
  VaComm1.PortNum := 1;
  VaComm1.BaudRate := br9600;           {Baudrate = 9 600}
  VaComm1.Databits := db8;              {Databits = 8}
  VaComm1.Parity := paNone;             {Parity = None}
  VaComm1.StopBits := sb1;              {Stopbits = 10}
  VaComm1.Open;                         {Open the comm port}
  Comm1Cts(VaComm1);
  Comm1Dsr(VaComm1);
  Comm1Ring(VaComm1);
  Comm1Rltd(VaComm1);
  for i := 1 to 995 do begin
    ch[i] := 'U';
  end;
  for i := 996 to 1000 do begin
    ch[i] := 'm';
  end;
  ch[1001] := 'M';
  ch[1002] := 'L';
  ch[1003] := 'S';
  i := 1003;

```



```
AssignFile(F1, 'c:\Mobrob Progs\Lydia Progs\Matlab Progs\current.pos');
```

```
Reset(F1);
```

```
while not Eof(F1) do begin
```

```
    inc(i);
```

```
    Read(F1, Ch[i]);
```

```
end;
```

```
CloseFile(F1);
```

```
inc(i);
```

```
Ch[i] := 'S';
```

```
Ch[i+1] := 'L';
```

```
Ch[i+2] := 'M';
```

```
filesize := i + 2;
```

```
for i := 1 to filesize do begin
```

```
    OK := VaComm1.WriteText(ch[i]);
```

```
    if not OK then
```

```
        Memo1.Lines.add('Error writing');
```

```
end;
```

```
for i := 1 to 10000 do begin
```

```
    counter := i;
```

```
end;
```

```
for i := 1 to 995 do begin
```

```
    ch[i] := 'U';
```

```
end;
```

```
for i := 996 to 1000 do begin
```

```
    ch[i] := 'm';
```

```
end;
```

```
ch[1001] := 'M';
```

```
ch[1002] := 'L';
```

```
ch[1003] := 'S';
i := 1003;

AssignFile(F1, 'c:\Mobrob Progs\Lydia Progs\Matlab Progs\wmap.map');
Reset(F1);
while not Eof(F1) do begin
    inc(i);
    Read(F1, Ch[i]);

end;

CloseFile(F1);

inc(i);
Ch[i] := 'S';
Ch[i+1] := 'L';
Ch[i+2] := 'M';

filesize := i + 2;

for i := 1 to filesize do begin
    OK := VaComm1.WriteText(ch[i]);
    if not OK then
        Memo1.Lines.add('Error writing');
end;

for i := 1 to 10000 do begin
    counter := i;
end;

for i := 1 to 995 do begin
    ch[i] := 'U';
end;
for i := 996 to 1000 do begin
    ch[i] := 'm';
```

```

end;
ch[1001] := 'M';
ch[1002] := 'L';
ch[1003] := 'S';
i := 1003;

AssignFile(F1, 'c:\Mobrob Progs\Lydia Progs\Matlab Progs\wmap.pth');
Reset(F1);
while not Eof(F1) do begin
    inc(i);
    Read(F1, Ch[i]);

end;

CloseFile(F1);

inc(i);
Ch[i] := 'S';
Ch[i+1] := 'L';
Ch[i+2] := 'M';

filesize := i + 2;

for i := 1 to filesize do begin
    OK := VaComm1.WriteText(ch[i]);
    if not OK then
        Memo1.Lines.add('Error writing');
end;

for i := 1 to 10000 do begin
    counter := i;
end;

VaComm1.Close;           {Close the comm port}
Comm1Cts(VaComm1);

```



```
Comm1Dsr(VaComm1);
Comm1Ring(VaComm1);
Comm1Rlsl(VaComm1);

{RemReceive.ShowModal;}
```

```
VaComm1.Open;
Comm1Cts(VaComm1);
Comm1Dsr(VaComm1);
Comm1Ring(VaComm1);
Comm1Rlsl(VaComm1);
```

```
i := 0;
```

```
repeat
```

```
  ch[1] := ch[2];
  ch[2] := ch[3];
```

```
repeat
```

```
  Ok := VaComm1.ReadChar(ch[3]);
```

```
until Ok
```

```
until (ch[1]='M') and (ch[2]='L') and (ch[3]='S');
```

```
repeat
```

```
  inc(i);
```

```
repeat
```

```
  Ok := VaComm1.ReadChar(ch[i]);
```

```
until Ok
```

```
until (ch[i-2]='S') and (ch[i-1]='L') and (ch[i]='M');
```

```
3  VaComm1.Close;
    Comm1Cts(VaComm1);
    Comm1Dsr(VaComm1);
    Comm1Ring(VaComm1);
    Comm1Rlsd(VaComm1);

    filesize1:=i-3;
    assignfile(F1,'c:\mobrob progs\lydia progs\matlab progs\current.pos');
    rewrite(F1);
    for Counter := 1 to filesize1-1 do begin
        write(F1,ch[counter]);
    end;

    closefile(F1);

end;
```

```
procedure TfrmMain.VaComm1Data(Sender: TObject; Count: Integer);
begin
    Memo2.Lines.Text := Memo2.Lines.Text + VaComm1.ReadText;
    Memo1.Lines.add('Reading ' + IntToStr(Count) + ' bytes');
end;
```

```
procedure TfrmMain.VaComm1Event(Sender: TObject);
begin
    Memo1.Lines.add('Event signal detected...');
end;
```

```
procedure TfrmMain.VaComm1Open(Sender: TObject);
```

```

begin
  Memo1.Lines.add('Port open');
end;

procedure TfrmMain.VaComm1Close(Sender: TObject);
begin
  Memo1.Lines.Add('Port closed');
end;

procedure TfrmMain.QPathPlanningClick(Sender: TObject);
begin
  WinExec((Pchar('c:\matlab\bin\matlab /r mainqp /nosplash /minimize')), SW_SHOW);
end;

procedure TfrmMain.PathPlanningClick(Sender: TObject);
begin
  WinExec((Pchar('c:\matlab\bin\matlab /r main /nosplash /minimize')), SW_SHOW);
end;

procedure TfrmMain.About1Click(Sender: TObject);
begin
  AboutBox.showmodal; {show about form as modal}
end;

end.

```

ABOUT.PAS

```

unit about;

interface

uses Windows, SysUtils, Classes, Graphics, Forms, Controls, StdCtrls,
  Buttons, ExtCtrls, ShellAPI;

```



```
type
```

```
  TAboutBox = class(TForm)
```

```
    Panel1: TPanel;
```

```
    ProgramIcon: TImage;
```

```
    ProductName: TLabel;
```

```
    Version: TLabel;
```

```
    OKButton: TButton;
```

```
    Memo1: TMemo;
```

```
    By: TLabel;
```

```
    URLblinf: TLabel;
```

```
    URLlbl: TLabel;
```

```
    procedure URLlblClick(Sender: TObject);
```

```
    procedure URLlblMouseMove(Sender: TObject; Shift: TShiftState; X,  
      Y: Integer);
```

```
    procedure Panel1MouseMove(Sender: TObject; Shift: TShiftState; X,  
      Y: Integer);
```

```
  private
```

```
    { Private declarations }
```

```
  public
```

```
    { Public declarations }
```

```
end;
```

```
var
```

```
  AboutBox: TAboutBox;
```

```
implementation
```

```
uses Filesend;
```

```
{ $R *.DFM }
```

```
procedure TAboutBox.URLlblClick(Sender: TObject);
```

```
begin
```

```
  ShellExecute(0,'open','http://www.bing.sun.ac.za/research/mobrob/mobrob.htm',Nil,Nil,SW_SHOWNORMAL); // got  
  o URL
```

```
end;
```

```
procedure TAboutBox.URLLblMouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
```

```
begin
```

```
  URLLbl.Font.Color := clRed;//change font color when mouse moved over it
```

```
end;
```

```
procedure TAboutBox.Panel1MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
```

```
begin
```

```
  URLLbl.Font.Color := clBlue;//change font color when mouse moved over it
```

```
end;
```

```
end.
```

REMINDERRECEIVE.PAS

```
unit ReminderReceive;
```

```
interface
```

```
uses
```

```
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;
```

```
type
```

```
  TRemSend = class(TForm)
```

```
    Label1: TLabel;
```

```
    SendOK: TButton;
```

```
    Label2: TLabel;
```

```
    Label3: TLabel;
```

```
    Label4: TLabel;
```

```
    procedure SendOKClick(Sender: TObject);
```

```
  private
```

```
    { Private declarations }
```

```
  public
```

```
    { Public declarations }
```

```
end;
```

```
var
```

```
  RemSend: TRemSend;
```

implementation

{ \$R *.DFM }

3 procedure TRemSend.SendOKClick(Sender: TObject);

begin

Remsend.Close;

end;

end.

REMINDERSEND.PAS

unit ReminderSend;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
StdCtrls;

type

TRemSend = class(TForm)

Label1: TLabel;

SendOK: TButton;

Label2: TLabel;

Label3: TLabel;

Label4: TLabel;

procedure SendOKClick(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

RemSend: TRemSend;

implementation


```
{R *.DFM}
```

```
procedure TRemSend.SendOKClick(Sender: TObject);
```

```
begin
```

```
  Remsend.Close;
```

```
end;
```

```
end.
```

APPENDIX F: MOBROB MATLAB PROGRAMS

The MATLAB code created and edited is included in this appendix.

MAIN.M

```
%Use superposition to calculate potential field
%pot = potmap(map);
%specpot = pot;
%save 'potvar' specpot
%load potvar;                %Load potential
cd 'c:\Mobrob Progs\Lydia Progs\Matlab Progs'
figure
hexp = 3;
hmant = 5;
hcmbutton=icontrol('style','pushbutton','string','Create Map','position',[0 450 80 20],'callback','[map, specpot, mapfile] = creatmap;');
hmbut=icontrol('style','pushbutton','string','Edit Map','position',[0 400 80 20],'callback','[map, specpot, mapfile] = editmap;');
hlmbut=icontrol('style','pushbutton','string','Load Map','position',[0 350 80 20],'callback','[map, specpot, mapfile, currentx, currenty, direction] = loadmap(currentx,currenty,direction);');
hstbut=icontrol('style','pushbutton','string','Set Target','position',[0 100 80 20],'callback','[currentx, currenty, direction] = target(map, direction, specpot,currentx, currenty);');
```

MAINQP.M

```
%Use superposition to calculate potential field
%pot = potmap(map);
%specpot = pot;
%save 'potvar' specpot
%load potvar;                %Load potential
cd 'c:\Mobrob Progs\Lydia Progs\Matlab Progs'
figure
[map, specpot, currentx, currenty, direction] = loadmapqp(currentx, currenty, direction);
hstbut=icontrol('style','pushbutton','string','Set Target','position',[0 300 80 20],'callback','[currentx, currenty, direction] = targetqp(map, currentx, currenty, specpot, direction);','visible','on');
```

CREAM.M

```

function newpathlog = cream(pathlog, map)
didchange = 1;
while (didchange == 1);
    didchange = 0;
    clear newpathlog;
    oindex = 1; % Old index
    nindex = 2; % New index
    newpathlog(1,:) = pathlog(1,:); % Starting point fixed
    while (oindex < (size(pathlog,1) - 1));
        localx(1) = pathlog(oindex, 1); % Local start x
        localy(1) = pathlog(oindex, 2); % Local start y
        localx(2) = pathlog(oindex + 2, 1); % Local end x
        localy(2) = pathlog(oindex + 2, 2); % Local end y
        minx = min(localx);
        maxx = max(localx);
        miny = min(localy);
        maxy = max(localy);
        obstacles = 0;
        for xindex = minx: maxx;
            for yindex = miny: maxy;
                obstacles = obstacles + map(yindex+1, xindex+1);
            end
        end
        if (obstacles == 0) % Adjust if no obstacles present
            newpathlog(nindex,:) = pathlog(oindex + 2,:); % Remove 1 point
            oindex = oindex + 2; % Skip removed point
            nindex = nindex + 1;
            didchange = 1;
        else
            newpathlog(nindex,:) = pathlog(oindex + 1,:); % Keep centre point
            oindex = oindex + 1; % Goto next point
            nindex = nindex + 1; % Obvious
        end
    end
end % end while

```



```

if (oindex == (size(pathlog,1) - 1));           % 2nd last already stored
    newpathlog(nindex,:) = pathlog(oindex + 1, :); % Store last point
end
pathlog = newpathlog;
end                                             % didchange while
return

```

CREATMAP.M

```

function [map, specpot, file] = creatmap
clear omap
clear map
clear specpot
mode = 0                                     %Initialise mode
hold off
plot(0,0)
promptx={'Length in m (maximum x)','Width in m (maximum y)'}
Title = 'Factory Dimensions'
mapsizes=inputdlg(promptx,Title)
fields = {'x','y'}
s = cell2struct(mapsizes,fields,1)
mapxsize = str2num(s.x)*5+1
mapysize = str2num(s.y)*5+1
mapxsize = floor(mapxsize)
mapysize = floor(mapysize)
for xindex = 1 : (mapxsize)
    for yindex = 1 : (mapysize)
        if (xindex == 1)
            map(yindex,xindex) = 1;
        elseif (xindex == (mapxsize))
            map(yindex,xindex) = 1;
        elseif (yindex == 1)
            map(yindex,xindex) = 1;
        elseif (yindex == (mapysize))
            map(yindex,xindex) = 1;
        else
            map(yindex,xindex) = 0;
        end
    end
end

```

```

        end
    end                % For yindex
end                    % For xindex
grid on
set(gca,'YDir','reverse')
3 for xindex = 1 : mapxsize;
    for yindex = 1 : mapysize;
        if map(yindex,xindex) == 0
            set(gca,'YDir','reverse')
            plot(0,0,(xindex-1)*0.2,(yindex-1)*0.2, 'b')
        else
            set(gca,'YDir','reverse')
            plot(0,0,(xindex-1)*0.2,(yindex-1)*0.2, 'b*')
        end
        hold on
    end
end
set(gca,'YDir','reverse')
plot(0,0,1.01*(mapxsize*0.2),1.01*(mapysize*0.2), 'r^')
grid on
while mode == 0
    [startx, starty, button] = ginput(1);
    % User inputs start coords
    if or(startx*5 > mapxsize, starty*5 > mapysize)
        mode = 1
    else
        startx = round(startx*5);
        starty = round(starty*5);
        if map(starty+1,startx+1) == 1
            map(starty+1,startx+1) = 0;
            set(gca,'YDir','reverse')
            plot(0,0,startx/5,starty/5, 'w*')
            plot(0,0,startx/5,starty/5, 'b')
        else
            map(starty+1,startx+1) = 1;
            set(gca,'YDir','reverse')
        end
    end
end

```

```

        plot(0,0,startx/5, starty/5, 'b*')
    end
end
end
specpot = potmap(map);
3 [fname, pname] = uiputfile('*.mop','Save Factory Map');           % Let user choose file
justname=strtok(fname, '.');
mapname = [pname, justname, '.mop']
potname = [pname, justname, '.pot']
save(mapname,'map','-ascii')
save(potname,'specpot','-ascii')
hold off
plot(0,0)
return

```

EDITMAP.M

```

function [map, specpot, file] = editmap
clear omap
clear map
clear pot
clear specpot
hold off
plot(0,0)
mode = 0
[fname, pname] = uigetfile('*.mop','Select Factory Map');           % Let user choose file
justname = strtok(fname, '.');                                       % Retrieve name part
% Load factory map
map = load([pname fname]);
%omap = eval([justname]);                                             % omap contains info
% Draw plan (only once)
[mapysize, mapxsize] = size(map);
grid on
set(gca,'YDir','reverse')
for xindex = 1 : mapxsize;
    for yindex = 1 : mapysize;
        if map(yindex,xindex) == 0

```



```

        set(gca,'YDir','reverse')
        plot(0,0,(xindex-1)*0.2,(yindex-1)*0.2, 'b')
    else
        set(gca,'YDir','reverse')
        plot(0,0,(xindex-1)*0.2,(yindex-1)*0.2, 'b*')
    end
    hold on
end

end
set(gca,'YDir','reverse')
plot(0,0,1.01*(mapxsize*0.2),1.01*(mapysize*0.2),'r^')
while mode == 0
    grid on
    [startx, starty, button] = ginput(1);
    % User inputs start coords
    if startx > mapxsize/5
        if starty > mapysize/5
            mode = 1
        end
    else
        startx = round(startx*5);
        starty = round(starty*5);
        if map(starty+1,startx+1) == 1
            map(starty+1,startx+1) = 0;
            set(gca,'YDir','reverse')
            plot(0,0,startx/5,starty/5, 'w*')
            plot(0,0,startx/5,starty/5, 'b')
        else
            map(starty+1,startx+1) = 1;
            set(gca,'YDir','reverse')
            plot(0,0,startx/5, starty/5, 'b*')
        end
    end
    grid on
end
end
specpot = potmap(map);

```

```

[fname, pname] = uiputfile('*.mop','Save Factory Map');           % Let user choose file
justname=strtok(fname, '.');
mapname = [pname, justname, '.mop']
potname = [pname, justname, '.pot']
save(mapname,'map','-ascii')
save(potname,'specpot','-ascii')
hold off
plot(0,0)
return

```

INISLIDE.M

```

function [hmant, hexp, hdir] = inislide
hmant = uicontrol('Style','slider','Min',1,'Max',9,'Position',[0 0 100 30], 'string', 'Weight')
hexp = uicontrol('Style','slider','Min',1,'Max',8,'Position',[200 0 100 30], 'string', '10^x')
hdir = uicontrol('Style','slider','Min',1,'Max',8,'Position',[400 0 100 30], 'string', 'Direction')
set(hmant, 'callback', showvals(hmant, hexp, hdir))
set(hexp, 'callback', showvals(hmant, hexp, hdir))
set(hdir, 'callback', showvals(hmant, hexp, hdir))

```

LOADMAP.M

```

function [map, specpot, file, currentx, currenty, direction] = loadmap(currentx,currenty,direction)
clear omap
clear map
clear pot
clear specpot
hold off
plot(0,0)
[pos,currentx,currenty,direction] = loadpos;
[fname, pname] = uigetfile('*.mop','Select Factory Map');           % Let user choose file
justname = strtok(fname, '.');                                     % Retrieve name part
% Load factory map
map = load([pname justname '.mop']);
%omap = eval([justname]);                                         % omap contains info
% Load potential file
%clear eval([justname])

```

```

specpot = load([pname justname '.pot']);
%specpot = eval([justname]);
file = [pname justname]
title(file);
% Draw plan (only once)
3 [mapysize, mapxsize] = size(map);
hold off
set(gca,'YDir','reverse')
plot(0,0,mapxsize/5,mapysize/5)
for xindex = 1 : mapxsize;
    for yindex = 1 : mapysize;
        hold on
        if sign(map(yindex, xindex)) == 1

            plot(0,0,(xindex-1)*0.2, (yindex-1)*0.2, 'b*')
        else
            plot(0,0,(xindex-1)*0.2, (yindex-1)*0.2, 'b')
        end
    end
end
end
plot(currentx/100,currenty/100,'g*')
set(gca,'YDir','reverse')
grid on
return

```

LOADMAPQP.M

```

function [map, specpot, currentx, currenty, direction] = loadmapqp(currentx, currenty, direction)
clear omap
clear map
clear pot
clear specpot
[pos,currentx,currenty,direction] = loadpos;
justname = 'c:\Mobrob Progs\Lydia Progs\Matlab Progs\wmap'; % Retrieve name part
% Load factory map
map = load([justname '.mop']);
% Load potential file

```



```

clear eval([justname])
specpot = load([justname '.pot']);
% Draw plan (only once)
[mapysize, mapxsize] = size(map);
hold off
set(gca,'YDir','reverse')
plot(0,0,mapxsize/5,mapysize/5)
for xindex = 1 : mapxsize;
    for yindex = 1 : mapysize;
        hold on
        if sign(map(yindex, xindex)) == 1

            plot(0,0,(xindex-1)*0.2, (yindex-1)*0.2, 'b*')
        else
            plot(0,0,(xindex-1)*0.2, (yindex-1)*0.2, 'b')
        end
    end
end
end
plot(currentx/100,currenty/100,'g*') %Currentx, currenty in cm
set(gca,'YDir','reverse')
grid on
return

```

LOADPOS.M

```

function [pos,currentx,currenty,direction] = loadmapqp
clear pos
justname = 'c:\Mobrob Progs\Lydia Progs\Matlab Progs\current';
pos = load([justname '.pos']);
%pos = eval([justname]);
currentx = pos(1);
currenty = pos(2);
direction = pos(3);
return

```

NEWMAP.M

```

function map = newmap(oldmap)
[mapysize, mapxsize] = size(oldmap)
map = 1 * ones(mapysize + 2, mapxsize + 2);
for xindex = 1 : mapxsize;
    for yindex = 1 : mapysize;
        map(yindex + 1, xindex + 1) = oldmap(yindex, xindex);
    end
end
return

```

NEWPOS.M

```

function [newposy, newposx, direction, stop] = newpos(posib, posy, posx, direction, map);
% Calculate new posy, posx from posib. and impmove
infinity = 1e5;
oposx = posx
oposy = posy
odirection = direction
deltax = [0 1 1 1 0 -1 -1 -1];
deltay = [-1 -1 0 1 1 1 0 -1];
impmove=[ 0 0 0 1 1 1 0 0;
          0 0 0 0 1 1 1 0;
          0 0 0 0 0 1 1 1;
          1 0 0 0 0 0 1 1;
          1 1 0 0 0 0 0 1;
          1 1 1 0 0 0 0 0;
          0 1 1 1 0 0 0 0;
          0 0 1 1 1 0 0 0;];
deltainf= infinity - posib;
posib = impmove(direction, :).* deltainf + posib;
total = sum(posib);
if direction == 8
    options = [posib(7),posib(8),posib(1)]
elseif direction == 1
    options = [posib(8),posib(1),posib(2)]
else
    options = [posib(direction-1),posib(direction),posib(direction+1)]

```

```

end
if( total>0.95*8*infinity*4)
    stop = 1
    newposx = posx
    newposy = posy
3 else
    [dummy, position] = min(options);

    if and(direction == 8,position(1) == 3)
        direction = 1
    elseif and(direction == 1,position(1) == 1)
        direction = 8
    else
        direction = direction + position(1) - 2
    end
    newposx = posx + deltax(direction);
    newposy = posy + deltay(direction);
    if map(newposy+1, newposx+1) == 1
        if map(oposy+1, oposx+1) == 1
            if newposx ~= oposx
                if newposy ~= oposy
                    if map(newposy+1, oposx+1) == 1
                        if map(oposy+1, newposx+1) == 1
                            direction = odirection
                            options(position) = options(position)+1000
                            [dummy, position] = min(options);
                            if and(direction == 8,position(1) == 3)
                                direction = 1
                            elseif and(direction == 1,position(1) == 1)
                                direction = 8
                            else
                                direction = direction + position(1) - 2
                            end
                            newposx = posx + deltax(direction);
                            newposy = posy + deltay(direction);
                        end
                    end
                end
            end
        end
    end
end

```



```

        end
    end
end
end
end
stop = 0;
end
return

```

POSIB.M

```

function nesw = posib(pot, posy, posx);
% nesw = posib(pot, posy, posx)
% Returns possiblity matrix
% N NE E SE S SW W NW
[mapysize, mapxsize] = size(pot);
nesw = ones(1,8);
thispot = pot(posy, posx);           % Potential at location
nesw(1) = pot(posy-1, posx) + pot(posy-2,posx-1) + pot(posy-2,posx) + pot(posy-2,posx+1);
nesw(2) = pot(posy-1, posx+1) + pot(posy-2,posx+1) + pot(posy-2,posx+2) + pot(posy-1,posx+2);
nesw(3) = pot(posy, posx+1) + pot(posy-1,posx+2) + pot(posy,posx+2) + pot(posy+1,posx+2);
nesw(4) = pot(posy+1, posx+1) + pot(posy+1,posx+2) + pot(posy+2,posx+2) + pot(posy+2,posx+1);
nesw(5) = pot(posy+1, posx) + pot(posy+2,posx+1) + pot(posy+2,posx) + pot(posy+2,posx-1);
nesw(6) = pot(posy+1, posx-1) + pot(posy+2,posx-1) + pot(posy+2,posx-2) + pot(posy+1,posx-2);
nesw(7) = pot(posy, posx-1) + pot(posy+1,posx-2) + pot(posy,posx-2) + pot(posy-1,posx-2);
nesw(8) = pot(posy-1, posx-1) + pot(posy-1,posx-2) + pot(posy-2,posx-2) + pot(posy-2,posx-1);
return
% North
if (posy ~=2)
    posib(1) = pot(posy-1, posx) - thispot;
end
% North-East
if( posx ~= mapxsize-1 & posy ~= 2)
    posib(2) = pot(posy-1 , posx+1) - thispot;
end
% East
if ( posx ~= mapxsize-1)

```

```

    posib(3) = pot(posy, posx+1) - thispot;
end
% South-East
if( posx ~= mapxsize-1 & posy ~= mapysize-1)
    posib(4) = pot(posy+1, posx+1) - thispot;
end
% South
if( posy ~= mapxsize-1 )
    posib(5) = pot(posy+1, posx) - thispot;
end
% South-West
if( posx ~= 2 & posy ~= mapysize-1)
    posib(6) = pot(posy+1, posx-1) - thispot;
end
% West
if(posx ~= 2)
    posib(7) = pot(posy, posx-1) - thispot;
end
% North-West
if( posx ~=2 & posy ~= 2)
    posib(8) = pot(posy-1, posx-1) - thispot;
end
nesw = posib;
return

```

POTMAP.M

```

function pot = potmap(map);
% Create Potential map from obstacle map.
infinity = 1e5;
[mapysize, mapxsize] = size(map);
pot = infinity * ones(mapysize, mapxsize);
determined
for potx = [1: mapxsize];
    for poty = [1: mapysize];
        if ( map(poty, potx) ~= 0)

```

% Get size from map
% All four walls defined as obstacles
% potx & poty contains coord of potential to be
% Assume scaled 'infinite'

potential at location of obstacle.

```

    partpot = map(poty,potx) * infinity;
else
    partpot = 0;
    for xindex = [1 : mapxsize];
        for yindex= [1: mapysize];
            r = sqrt((xindex - potx)^2 + (yindex - poty)^2);
            if(r~=0) % Do not include own
location in calcs.
                partpot = partpot + (map(yindex, xindex)/r);
            end
        end
    end
    pot(poty, potx) = partpot;
end
end
return

```

POTMAP2.M

```

function pot = potmap2(map, pot, targetx, targety, weight);
% Calculate potential due to single point at (targetx, targety) with weight weight.
infinity = 1e5;
[mapysize, mapxsize] = size(map); % Get size from potential map
for potx = [2 : mapxsize-1]; % potx & poty contains coord of potential to be determined
    for poty = [2 : mapysize-1];
        if (map(poty, potx) == 0)
            r = sqrt( (targetx - potx)^2 + (targety - poty)^2);
            if(r~=0) % Do not include own location in calcs.
                pot(poty, potx) = pot(poty, potx) + weight/r;
            end
        end
    end
end
pot(targety, targetx) = weight * infinity;
return

```


STARGET.M

```
function[currentx, currenty, direction] = starget(map, direction, specpot, currentx,currenty)
Ok = 0;
targetx = 1;
targety = 1;
while (Ok == 0);
    [targetx, targety] = ginput(1);
    targetx2 = round(targetx*5);
    targety2 = round(targety*5);
    if map(targety2, targetx2) == 0
        Ok = 1;
    end;
end
mapname = 'c:\Mobrob Progs\Lydia Progs\Matlab Progs\wmap.mop'
potname = 'c:\Mobrob Progs\Lydia Progs\Matlab Progs\wmap.pot'
save(mapname,'map','-ascii')           %Save files for Quick planning
save(potname,'specpot','-ascii')       %Save files for Quick planning
plot(targetx2/5, targety2/5, 'r*')
walk(map, specpot, targetx2, targety2)
return
```

STARGETQP.M

```
function [currentx, currenty, direction] = stargetqp(map, currentx, currenty, specpot, direction)
Ok = 0;
plot(currentx/100, currenty/100, 'g*')
targetx = 1;
targety = 1;
while (Ok == 0);
    [targetx, targety] = ginput(1);
    targetx2 = round(targetx*5);
    targety2 = round(targety*5);
    if map(targety2, targetx2) == 0
        Ok = 1;
    end;
end
```

```

plot(targetx2/5, targety2/5, 'r*')
walkqp(map,specpot,targetx2,targety2);
return

```

WALK.M

```

function walk(map,specpot,targetx,targety)
[pos,currentx,currenty,direction] = loadpos;
startx = currentx/100;
starty = currenty/100;
plot(startx,starty,'g*')
tweight = -(10^3) * 5;
pot = potmap2(map, specpot, targetx, targety, tweight);
posx = startx*5;
posy = starty*5;
posx, posy, direction, tweight
pathlog = [posx posy direction];
impmove = zeros(1,8);
strike = 0; % Not on target.... yet! :-)
num = 2
while strike == 0
    oposy = posy;
    oposx = posx;
    nesw = posib(pot, posy, posx);
    [posy, posx, direction, strike] = newpos(nesw, posy, posx, direction, map);
    hold on
    line([oposx/5 posx/5], [oposy/5 posy/5], 'Color', [1 1 1]);
    if ( posx == targetx & posy == targety)
        strike = 1;
    end
    pathlog(num,1) = posx;
    pathlog(num,2) = posy;
    pathlog(num,3) = direction;
    num = num + 1
end % while strike end
[index,dummy] = size(pathlog)
%for counter = 4:index

```

```

% newpathlog(counter-3,1) = pathlog(counter,1);
% newpathlog(counter-3,2) = pathlog(counter,2);
%end
%for counter = 1:3
% newpathlog2(counter,1) = pathlog(counter,1);
% newpathlog2(counter,2) = pathlog(counter,2);
%end

for counter = 1:index
    newpathlog(counter,1) = pathlog(counter,1);
    newpathlog(counter,2) = pathlog(counter,2);
end
newpathlog = cream(newpathlog, map);
%newpathlog = [newpathlog2; newpathlog]
hold on
plot (newpathlog(:,1)/5,newpathlog(:,2)/5,'r');
%plot (pathlog(:,1),pathlog(:,2),'r');
%newpathlog = pathlog
[fname, pname] = uiputfile('*.pth','Save Factory Path'); % Let user choose file
justname=strtok(fname,'');
dimentions = size(newpathlog); % Determine the number of nodes
nodes = dimentions(1);
[mapysize, mapxsize] = size(map);
mapxsize = mapxsize - 1;
mapysize = mapysize - 1;

First1 = ['MapFile ' justname];
First2 = ['MaxNode ' num2str(nodes)];
First3 = ['Max X ' num2str(mapxsize)];
First4 = ['Max Y ' num2str(mapysize)]

pathname = [pname, justname, '.pth']
mapname = [pname, justname, '.map']

for index = 1:nodes

```



```

path(index,1) = index;

fmap(index,1) = index-1;
fmap(index,2) = newpathlog(index,2)*20;           %grid of 20cmx20cm
fmap(index,3) = newpathlog(index,1)*20;

if index == 1
    fmap(index,4) = 1;
    fmap(index,5) = 1;
elseif index == nodes
    fmap(index,4) = 1;
    fmap(index,5) = (index - 2);
else
    fmap(index,4) = 2;
    fmap(index,5) = (index - 2);
    fmap(index,6) = (index);
end
end

fid = fopen(pathname,'w');
for index = 1 : (nodes-1)
    fprintf(fid,'%1.0f\n',path(index,1))
end
fprintf(fid,'END')
fclose(fid)
Newstr = ['MapFile ',justname,'\n']

MaxX = mapxsize*10;
MaxY = mapysize*10;
fid = fopen(mapname,'w');
fprintf(fid,Newstr)
fprintf(fid,'MaxNode %1.0f\n',nodes-1)
fprintf(fid,'Max X   %1.0f\n',(mapysize)*20)
fprintf(fid,'Max Y   %1.0f\n',(mapxsize)*20)
fprintf(fid,'NodeNR  Xpos   Ypos   NrLink  LNdeNr1 LNdeNr2 LNdeNr3 LNdeNr4 LNdeNr5 LNdeNr6 \n')
fprintf(fid,'1      2      3      4      5      6      7      8      9      10      Column Number \n')
[lines, columns] = size(fmap);

```

```

a = 0
b = 0
for index = 1:lines
    test = 0
    for index2 = 1:columns
        if index2~=1
            if fmap(index,index2-1)<10
                a = 1
            elseif fmap(index,index2-1)<100
                a = 2
            elseif fmap(index,index2-1)<1000
                a = 3
            elseif fmap(index,index2-1)<10000
                a = 4
            elseif fmap(index,index2-1)<100000
                a = 5
            elseif fmap(index,index2-1)<1000000
                a = 6
            else a = 7
            end

            if fmap(index,index2)<10
                b = 1
            elseif fmap(index,index2)<100
                b = 2
            elseif fmap(index,index2)<1000
                b = 3
            elseif fmap(index,index2)<10000
                b = 4
            elseif fmap(index,index2)<100000
                b = 5
            elseif fmap(index,index2)<1000000
                b = 6
            else b = 7
            end
        end
    end
end

```

```

if index2 == 1
    fprintf(fid,'%1.0f',fmap(index,index2));

elseif (index2 == columns)
    if (fmap(index,index2) ~= 0)
        if test == 0                                %Test for eol test = 1 true
            if (8-a+b) == 14
                fprintf(fid,'%14.0f\n',fmap(index,index2));
            end
            if (8-a+b) == 13
                fprintf(fid,'%13.0f\n',fmap(index,index2));
            end

            if (8-a+b) == 12
                fprintf(fid,'%12.0f\n',fmap(index,index2));
            end

            if (8-a+b) == 11
                fprintf(fid,'%11.0f\n',fmap(index,index2));
            end

            if (8-a+b) == 10
                fprintf(fid,'%10.0f\n',fmap(index,index2));
            end

            if (8-a+b) == 9
                fprintf(fid,'%9.0f\n',fmap(index,index2));
            end

            if (8-a+b) == 8
                fprintf(fid,'%8.0f\n',fmap(index,index2));
            end

            if (8-a+b) == 7
                fprintf(fid,'%7.0f\n',fmap(index,index2));

```



```

end

if (8-a+b) == 6
    fprintf(fid,'%6.0f\n',fmap(index,index2));
end

if (8-a+b) == 5
    fprintf(fid,'%5.0f\n',fmap(index,index2));
end

if (8-a+b) == 4
    fprintf(fid,'%4.0f\n',fmap(index,index2));
end

if (8-a+b) == 3
    fprintf(fid,'%3.0f\n',fmap(index,index2));
end

if (8-a+b) == 2
    fprintf(fid,'%2.0f\n',fmap(index,index2));
end
end
else
    fprintf(fid,'\n');
    test = 0                %Reset test
end
elseif fmap(index,index2) == 0
    if fmap(index,index2+1) ~= 0    %Next ~= 0
        if (8-a+b) == 14
            fprintf(fid,'%14.0f',fmap(index,index2));
        end

        if (8-a+b) == 13
            fprintf(fid,'%13.0f',fmap(index,index2));
        end
    end
end

```

```
if (8-a+b) == 12
    fprintf(fid,'%12.0f',fmap(index,index2));
end
```

```
if (8-a+b) == 11
    fprintf(fid,'%11.0f',fmap(index,index2));
end
```

```
if (8-a+b) == 10
    fprintf(fid,'%10.0f',fmap(index,index2));
end
```

```
if (8-a+b) == 9
    fprintf(fid,'%9.0f',fmap(index,index2));
end
```

```
if (8-a+b) == 8
    fprintf(fid,'%8.0f',fmap(index,index2));
end
```

```
if (8-a+b) == 7
    fprintf(fid,'%7.0f',fmap(index,index2));
end
```

```
if (8-a+b) == 6
    fprintf(fid,'%6.0f',fmap(index,index2));
end
```

```
if (8-a+b) == 5
    fprintf(fid,'%5.0f',fmap(index,index2));
end
```

```
if (8-a+b) == 4
    fprintf(fid,'%4.0f',fmap(index,index2));
end
```

```
if (8-a+b) == 3
    fprintf(fid,'%3.0f',fmap(index,index2));
end

if (8-a+b) == 2
    fprintf(fid,'%2.0f',fmap(index,index2));
end

else
    fprintf(fid,'\n');
    test = 1
end

elseif index2 ~=1
    if (8-a+b) == 14
        fprintf(fid,'%14.0f',fmap(index,index2));
    end

    if (8-a+b) == 13
        fprintf(fid,'%13.0f',fmap(index,index2));
    end

    if (8-a+b) == 12
        fprintf(fid,'%12.0f',fmap(index,index2));
    end

    if (8-a+b) == 11
        fprintf(fid,'%11.0f',fmap(index,index2));
    end

    if (8-a+b) == 10
        fprintf(fid,'%10.0f',fmap(index,index2));
    end

    if (8-a+b) == 9
```



```
        fprintf(fid,'%9.0f',fmap(index,index2));
    end

    if (8-a+b) == 8
        fprintf(fid,'%8.0f',fmap(index,index2));
    end

    if (8-a+b) == 7
        fprintf(fid,'%7.0f',fmap(index,index2));
    end

    if (8-a+b) == 6
        fprintf(fid,'%6.0f',fmap(index,index2));
    end

    if (8-a+b) == 5
        fprintf(fid,'%5.0f',fmap(index,index2));
    end

    if (8-a+b) == 4
        fprintf(fid,'%4.0f',fmap(index,index2));
    end

    if (8-a+b) == 3
        fprintf(fid,'%3.0f',fmap(index,index2));
    end

    if (8-a+b) == 2
        fprintf(fid,'%2.0f',fmap(index,index2));
    end

end

end

end
```

```
fclose(fid)
```

```
return
```

WALKQP.M

```
function walkqp(map,specpot,targetx,targety)
```

```
    [pos,currentx,currenty,direction] = loadpos;
```

```
    startx = currentx/100;
```

```
    starty = currenty/100;
```

```
    plot(startx,starty,'g*')
```

```
    tweight = -5000;
```

```
    pot = potmap2(map, specpot, targetx, targety, tweight);
```

```
    posx = startx*5;
```

```
    posy = starty*5;
```

```
    posx, posy, direction, tweight
```

```
    pathlog = [posx posy direction];
```

```
    impmove = zeros(1,8);
```

```
    strike = 0;
```

```
    % Not on target.... yet! :-)
```

```
    num = 2
```

```
    while strike == 0
```

```
        oposy = posy;
```

```
        oposx = posx;
```

```
        nesw = posib(pot, posy, posx);
```

```
        [posy, posx, direction, strike] = newpos(nesw, posy, posx, direction, map);
```

```
        hold on
```

```
        line([oposx/5 posx/5], [oposy/5 posy/5], 'Color',[1 1 1]);
```

```
        if ( posx == targetx & posy == targety)
```

```
            strike = 1;
```

```
        end
```

```
        pathlog(num,1) = posx;
```

```
        pathlog(num,2) = posy;
```

```
        pathlog(num,3) = direction;
```

```
        num = num + 1
```

```

end                                     % while strike end

[index,dummy] = size(pathlog)

%for counter = 4:index
% newpathlog(counter-3,1) = pathlog(counter,1);
% newpathlog(counter-3,2) = pathlog(counter,2);
%end

%for counter = 1:3
% newpathlog2(counter,1) = pathlog(counter,1);
% newpathlog2(counter,2) = pathlog(counter,2);
%end

for counter = 1:index
    newpathlog(counter,1) = pathlog(counter,1);
    newpathlog(counter,2) = pathlog(counter,2);
end

newpathlog = cream(newpathlog, map);
%newpathlog = [newpathlog2; newpathlog]
hold on
plot (newpathlog(:,1)/5,newpathlog(:,2)/5,'r');
%plot (pathlog(:,1),pathlog(:,2),'r');
%newpathlog = pathlog

justname='c:\Mobrob Progs\Lydia Progs\Matlab Progs\wmap';

dimentions = size(newpathlog);          % Determine the number of nodes
nodes = dimentions(1);

[mapysize, mapxsize] = size(map);
mapxsize = mapxsize - 1;
mapysize = mapysize - 1;

pathname = [justname, '.pth']

```



```

mapname = [justname, '.map']

for index = 1:nodes

    path(index,1) = index;

    fmap(index,1) = index-1;
    fmap(index,2) = newpathlog(index,2)*20;           %grid of 20cmx20cm
    fmap(index,3) = newpathlog(index,1)*20;
    if index == 1
        fmap(index,4) = 1;
        fmap(index,5) = 1;
    elseif index == nodes
        fmap(index,4) = 1;
        fmap(index,5) = (index - 2);
    else
        fmap(index,4) = 2;
        fmap(index,5) = (index - 2);
        fmap(index,6) = (index);
    end

end

fid = fopen(pathname,'w');
for index = 1 : (nodes-1)
    fprintf(fid,'%1.0f\n',path(index,1))
end
fprintf(fid,'END')
fclose(fid)

MaxX = mapxsize*10;
MaxY = mapysize*10;
fid = fopen(mapname,'w');

```

```

fprintf(fid,'MapFile wmap \n')
fprintf(fid,'MaxNode %1.0f\n',nodes-1)
fprintf(fid,'Max X  %1.0f\n',(mapysize)*20)
fprintf(fid,'Max Y  %1.0f\n',(mapxsize)*20)
fprintf(fid,'NodeNR Xpos  Ypos  NrLink  LNdeNr1 LNdeNr2 LNdeNr3 LNdeNr4 LNdeNr5 LNdeNr6 \n')
fprintf(fid,'1    2    3    4    5    6    7    5    9    10    Column Number \n')
[lines, columns] = size(fmap);
a = 0
b = 0
for index = 1:lines
    test = 0
    for index2 = 1:columns
        if index2~=1
            if fmap(index,index2-1)<10
                a = 1
            elseif fmap(index,index2-1)<100
                a = 2
            elseif fmap(index,index2-1)<1000
                a = 3
            elseif fmap(index,index2-1)<10000
                a = 4
            elseif fmap(index,index2-1)<100000
                a = 5
            elseif fmap(index,index2-1)<1000000
                a = 6
            else a = 7
        end

        if fmap(index,index2)<10
            b = 1
        elseif fmap(index,index2)<100
            b = 2
        elseif fmap(index,index2)<1000
            b = 3
        elseif fmap(index,index2)<10000
            b = 4

```

```

elseif fmap(index,index2)<100000
    b = 5
elseif fmap(index,index2)<1000000
    b = 6
else b = 7
end
end

if index2 == 1
    fprintf(fid,'%1.0f',fmap(index,index2));

elseif (index2 == columns)
    if (fmap(index,index2) ~= 0)
        if test == 0                %Test for eol test = 1 true
            if (8-a+b) == 14
                fprintf(fid,'%14.0f\n',fmap(index,index2));
            end

            if (8-a+b) == 13
                fprintf(fid,'%13.0f\n',fmap(index,index2));
            end

            if (8-a+b) == 12
                fprintf(fid,'%12.0f\n',fmap(index,index2));
            end

            if (8-a+b) == 11
                fprintf(fid,'%11.0f\n',fmap(index,index2));
            end

            if (8-a+b) == 10
                fprintf(fid,'%10.0f\n',fmap(index,index2));
            end

            if (8-a+b) == 9

```



```

        fprintf(fid,'%9.0f\n',fmap(index,index2));
    end

    if (8-a+b) == 8
        fprintf(fid,'%8.0f\n',fmap(index,index2));
    end

    if (8-a+b) == 7
        fprintf(fid,'%7.0f\n',fmap(index,index2));
    end

    if (8-a+b) == 6
        fprintf(fid,'%6.0f\n',fmap(index,index2));
    end

    if (8-a+b) == 5
        fprintf(fid,'%5.0f\n',fmap(index,index2));
    end

    if (8-a+b) == 4
        fprintf(fid,'%4.0f\n',fmap(index,index2));
    end

    if (8-a+b) == 3
        fprintf(fid,'%3.0f\n',fmap(index,index2));
    end

    if (8-a+b) == 2
        fprintf(fid,'%2.0f\n',fmap(index,index2));
    end
end
else
    fprintf(fid,'\n');
    test = 0           %Reset test
end
elseif fmap(index,index2) == 0

```

```

if fmap(index,index2+1) ~= 0      %Next ~= 0
    if (8-a+b) == 14
        fprintf(fid,'%14.0f',fmap(index,index2));
    end

    if (8-a+b) == 13
        fprintf(fid,'%13.0f',fmap(index,index2));
    end

    if (8-a+b) == 12
        fprintf(fid,'%12.0f',fmap(index,index2));
    end

    if (8-a+b) == 11
        fprintf(fid,'%11.0f',fmap(index,index2));
    end

    if (8-a+b) == 10
        fprintf(fid,'%10.0f',fmap(index,index2));
    end

    if (8-a+b) == 9
        fprintf(fid,'%9.0f',fmap(index,index2));
    end

    if (8-a+b) == 8
        fprintf(fid,'%8.0f',fmap(index,index2));
    end

    if (8-a+b) == 7
        fprintf(fid,'%7.0f',fmap(index,index2));
    end

    if (8-a+b) == 6
        fprintf(fid,'%6.0f',fmap(index,index2));
    end

```

```

    if (8-a+b) == 5
        fprintf(fid,'%5.0f',fmap(index,index2));
    end

    if (8-a+b) == 4
        fprintf(fid,'%4.0f',fmap(index,index2));
    end

    if (8-a+b) == 3
        fprintf(fid,'%3.0f',fmap(index,index2));
    end

    if (8-a+b) == 2
        fprintf(fid,'%2.0f',fmap(index,index2));
    end

else
    fprintf(fid,'\n');
    test = 1
end

elseif index2 ~=1
    if (8-a+b) == 14
        fprintf(fid,'%14.0f',fmap(index,index2));
    end

    if (8-a+b) == 13
        fprintf(fid,'%13.0f',fmap(index,index2));
    end

    if (8-a+b) == 12
        fprintf(fid,'%12.0f',fmap(index,index2));
    end

```



```
if (8-a+b) == 11
    fprintf(fid,'%11.0f',fmap(index,index2));
end
```

```
if (8-a+b) == 10
    fprintf(fid,'%10.0f',fmap(index,index2));
end
```

```
if (8-a+b) == 9
    fprintf(fid,'%9.0f',fmap(index,index2));
end
```

```
if (8-a+b) == 8
    fprintf(fid,'%8.0f',fmap(index,index2));
end
```

```
if (8-a+b) == 7
    fprintf(fid,'%7.0f',fmap(index,index2));
end
```

```
if (8-a+b) == 6
    fprintf(fid,'%6.0f',fmap(index,index2));
end
```

```
if (8-a+b) == 5
    fprintf(fid,'%5.0f',fmap(index,index2));
end
```

```
if (8-a+b) == 4
    fprintf(fid,'%4.0f',fmap(index,index2));
end
```

```
if (8-a+b) == 3
    fprintf(fid,'%3.0f',fmap(index,index2));
end
```

```
        if (8-a+b) == 2
            fprintf(fid,'%2.0f',fmap(index,index2));
        end
    end
end
end
fclose(fid)
end
```